TREBAJO FIN DE CARRERA

TÍTOL DEL TFC: Creación de una interfaz gráfica de traducción automática para las lenguas de signos.

TITULACIÓ: Ingeniería Técnica de Telecomunicación, especialidad Sistemas de Telecomunicación.

AUTOR: Javier Marín Rey

DIRECTOR: Dolors Royo

SUPERVISOR: Guillem Massó (Barcelona Media)

DATA:

Títol: Creación de una interfaz gráfica de traducción automática para las lenguas de signos.
Autor: Javier Marín Rey
Director: Dolors Royo
Data:

Resumen

Con la realización de este trabajo se pretende el desarrollo y evaluación de una plataforma gráfica para el sistema de traducción automática Moses, que permite transformar frases de una lengua a otra, en nuestro caso, de catalán a Lengua de Signos Catalana. El sistema Moses es un software de código abierto que obtiene un modelo de traducción a partir del alineamiento y extracción de subsecuencias utilizando un corpus paralelo, compuesto por documentos de texto en la lengua origen y sus equivalentes en la lengua destino.

La plataforma está formado por 3 módulos principales: una aplicación cliente, una aplicación servidor y un avatar. La aplicación cliente envía un texto en catalán a la aplicación servidor, este ejecuta el sistema Moses para encontrar su traducción y busca en nuestra base de datos sus signos correspondientes codificados en formato XML mediante el sistema de notación HamNoSys para las lenguas de signos. Esta codificación es enviada al cliente para ser procesada por el avatar y ejecutar los movimientos correspondientes.

Por otro lado, hemos realizado un editor de signos que nos permite crear y editar signos y ser guardados luego a la base de datos. Todas estas aplicaciones han sido creadas en lenguaje Java.

Este sistema ayudará a mejorar la comunicación con las personas sordas signantes y tener el derecho de recibir la información en su lengua propia. La importancia de esta plataforma reside en la necesidad cada vez mayor de una herramienta que permita una traducción rápida y relativamente precisa entre lenguas.

El proyecto ha sido desarrollado con colaboración de Barcelona Media-Centro de Innovación.

Title: Creación de una interfaz gráfica de traducción automática para las lenguas de signos.
Author: Javier Marín Rey
Director: Dolors Royo
Date:

Overview

With the completion of this work aims at the development and evaluation of a graphical platform for the Moses machine translation system, which transforms phrases from one language to another, in our case, Catalan Catalan Sign Language. Moses is a system open source software gets a translation model from the alignment and extraction of subsequences using a parallel corpus, consisting of text documents in the source language and their equivalents in the target language.

The platform consists of 3 main modules: a client application, an application server and an avatar. The client application sends a text in Catalan to the application server which runs the system to find its translation Moses and search our database for the signs encoded in XML format by HamNoSys notation system for sign languages. This encoding is sent to the client to be processed by the avatar and run the corresponding movements.

On the other hand, we made a sign editor that lets you create and edit signs and then be stored in the database. All these applications have been created in Java.

This system will help improve communication with deaf signers and have the right to receive information in their own language. The importance of this platform lies in the increasing need for a tool that allows quick and relatively accurate translation between languages.

The project has been developed in collaboration with Barcelona Media-Center of Innovation.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

GLOSARIO

ASL	American Sign Language
BLEU	Bilingual Evaluation Understudy
eSIGN	Essential Sign Language Information on Government Networks
EUA	University of East Anglia
GIZA++	Herramienta para la traducción automática estadísitca
HamNoSys	Hamburg Sign Language Notation
IDE	Integrated Development Environment
IP	Internet Protocol
JAR	Java Archive
Java	Lenguaje de programación
Java SE	Java Standard Edition
JDBC	Java DataBase Connectivity
JDK	Java Development Kit
JNLP	Java Networking Launching Protocol
JRE	Java Runtime Environment
JVM	Java Virtual Machine
LGPL	Lesser General Publi License
LS	Lengua de Signos
LSC	Lengua de Signos Catalana
LSE	Lengua de Signos Española
MERT	Minimum Error Rate Training
MySQL	Sistema de gestión de base de datos
NIST	National Institute of Standards and Technology
SiGML	Signing Gesture Markup Language
SMT	Statistical Machine Translation
SRILM	SRI Language Modeling Toolkit
TA	Traducción Automática
TCP	Transmission Control Protocol
ViSiCAST	Virtual Signing, Captute, Animation, Storage and Transmission
XAMPP	X, Apache, MySQL, Php, Perl,
XML	Extensible Markup Language

INTRODUCCIÓN

Las personas sordas signantes, al igual que las personas oyentes, tienen el derecho de comunicarse y recibir información en su lengua propia. La lengua de signos es el modo de expresión utilizado por estas personas.

La comunidad sorda le ha costado mucho conseguir el reconocimiento social de su lengua el hecho de tratarse de una lengua que no utiliza como medio de transmisión en canal vocal.

La finalidad de este trabajo pretende desarrollar una interfaz gráfica de traducción automática para las lenguas de signos integrando por medio un avatar que ejecute los movimientos. Concretamente, traduciremos a la Lengua de Signos Catalana (LSC) a partir de un texto en catalán. Por otro lado desarrollaremos un editor de signos para crear o modificar signos y ver sus resultados por medio del dicho Avatar.

Para el desarrollo de este proyecto hemos creado dos aplicaciones (Cliente y Servidor) en Java, el cual un Cliente envía al Servidor un texto en catalán y recibe una trama de datos con la traducción del texto en una secuencia de signos (glosas) y su codificación equivalente en SiGML (Signing Gesture Markup Language). Esta codificación es un tipo de lenguaje del formato XML que permite procesar correctamente los movimientos de los signos por el Avatar JASigning, desarrollado en Europa para el proyecto eSiGN.

Para la traducción de textos, hemos integrado el sistema Moses en nuestra aplicación Servidor. Moses es un sistema de Traducción Automática Estadística que permite entrenar de forma automática modelos de traducción para cualquier par de lenguas. Únicamente se necesita una colección de textos traducidos (corpus paralelo). Un servidor de base de datos (MySQL) recogerá la secuencia de signos obtenidos por el sistema Moses y entregará su codificación en SiGML.

Desde Barcelona Media hemos entrenado un corpus paralelo con unas 260 frases relacionas en el ámbito de la meteorología y hemos preparado una base de datos de signos codificados en formato SiGML mediante el sistema de notación HamNoSys.

Por otro lado, hemos desarrollado un editor de signos en Java que permite crear o modificar un signo y ser ejecutado por el Avatar.

Con la llegada de los sistemas de traducción automática (TA) pueden convertirse en una herramienta muy útil para traducir textos en lenguas de signos. Hoy en día, la traducción es el principal cuello de botella de la sociedad de la información y su mecanización supone un gran adelanto en frente del alud de información y la necesidad de comunicación interlingüística.

La traducción automática se ha revelado útil en determinadas situaciones, especialmente en momentos de urgencia. Los programas de traducción automática

reducen costes y ahorran tiempo al traductor profesional para conseguir unos resultados aceptables en textos rutinarios y sencillos. Esta posibilidad convertiría a la traducción automática en una alternativa muy atractiva para investigadores científicos y técnicos.

Dicho esto, esta memoria está organizada en cinco capítulos repartidos de la siguiente manera:

En el capítulo 1 se pretende explicar que es la lengua de signos, su situación actual en la sociedad, sus características más importantes y como transcribir los signos de manera gráfica o textual.

En el capítulo 2 se centra el estudio y funcionamiento del sistema Moses. Previamente definiremos que es la traducción automática, sus características, ventajas e inconvenientes de cada tipo de traducción automática que podemos encontrar.

En el capítulo 3 trataremos de explicar como será el diseño del sistema, la arquitectura que se ha llevado a cabo, las herramientas de desarrollo empleadas, los diagramas de caso de uso para cada aplicación y el diseño de la base de datos.

En el capítulo 4 veremos el desarrollo de las aplicaciones Servidor, Cliente y Editor, comentando en cada aplicación las funciones que realiza y la estructura de las clases creadas.

Finalmente, el capítulo 5 comentaremos las pruebas realizadas experimentalmente, las limitaciones que hemos tenido a lo largo del proyecto, las posibles líneas futuras y las conclusiones personales.

CAPÍTULO 1. LA LENGUA DE SIGNOS

1.1. Introducción

Las lenguas de signos, al igual que las orales, se organizan por unidades elementales sin significado autónomo que, enlazadas, sirven para discutir sobre cualquier tema, desde lo más sencillo y concreto hasta lo más abstracto y complejo.

Así como las lenguas orales utilizan la vía vocal y auditiva para su producción y percepción respectivamente, la lengua de signos se produce mediante la vía gestual y se percibe mediante la vía visual. Es, por tanto, una lengua muy diferente en cuanto a su modo de producción y de percepción, por lo que requiere habilidades específicas para su aprendizaje, como son la atención, la discriminación visual y la agilidad manual.

Se utilizan las manos, y con elementos no manuales como movimientos de los labios, músculos faciales, acciones de la lengua, de los hombros y la cabeza. Estos elementos juegan un papel fundamental y hay que prestarles mucha atención porque son tan importantes como la acción que realizan las manos.

Todas las lenguas, orales o signadas, están compuestas por una serie de palabras o signos, que representan a algo y que combinadas de una determinada manera forman frases. En las lenguas orales el elemento básico son los sonidos, los fonemas, y combinados forman las palabras y con ellas las frases; y en las lenguas de signos el elemento básico son los queremas o parámetros formacionales, y con ellos se crean los signos y, a su vez, las frases signadas. Los queremas se agrupan en diferentes categorías: configuración (forma de la mano), localización, movimiento, orientación, dirección y expresión facial.

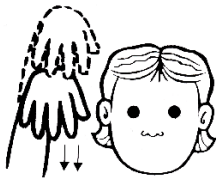


Fig. 1.1 Signo “PLUJA” de la Lengua de Signos Catalana

1.2. Situación actual

Según los estudios de diferentes fundaciones de la comunidad sorda, la cifra de personas sordas en España ronda el millón de personas. Se estima que el número de usuarios/as de la Lengua de Signos en España supera las 400.000 personas. Entre estos usuarios/as no sólo figuran las personas sordas sino todas aquellas que por razones familiares, afectivas o laborales han aprendido dicha lengua.

En nuestro país hay aproximadamente 500 Intérpretes de Lengua de Signos acreditados oficialmente que realizan alrededor de 40.000 servicios anuales. Los ámbitos y situaciones donde trabajan estos profesionales son tan diversos como los obstáculos y barreras de comunicación que encuentran las personas sordas: desde los centros de enseñanza, hasta las administraciones públicas, pasando por hospitales, comisarías, juzgados o espacios culturales.

Sin embargo, España está aún muy lejos de alcanzar la media europea en lo que a servicios de interpretación se refiere. Mientras que otros países europeos, hay un/a Intérprete por cada diez personas sordas, en nuestro país la proporción es de un/a Intérprete por cada 221 personas.

Contrariamente a lo que frecuentemente se cree, la lengua de signos no es universal y existen diferentes lenguas de signos en todo el mundo. Igual que sucede con las lenguas orales, a menudo las fronteras estatales no coinciden con las fronteras lingüísticas.

En España existen dos variantes: la Lengua de Signos Española (LSE) y la Lengua de Signos Catalana (LSC).

La existencia de la LSC se remonta a más de dos siglos de trayectoria, aunque a lo largo de la historia se la ha conocido con diferentes nombres (gestos, mímica, lenguaje mímico, lenguaje de signos, etc.).

La LSC cuenta con unos 25.000 usuarios, de los cuales unos 12.000 son personas sordas que optaron libremente por esta lengua. El resto son personas oyentes implicadas en la Comunidad Sorda en diferentes niveles: familiares, intérpretes de LS, profesionales cuya intervención se desarrolla en diferentes disciplinas educativas (educadores, logopedas, pedagogos, etc.), además del creciente número de personas oyentes que, sin tener ninguna afinidad, aprenden Lengua de Signos en las diferentes asociaciones de personas sordas.

1.3. Estudio lingüístico

La lingüística la podemos definir como la ciencia que estudia el lenguaje y las lenguas. Trata de explicar las estructuras que organizan las lenguas y las leyes que gobierna dichas estructuras.

En la lengua de signos podemos etiquetar las frases mediante glosas. Las glosas van especificadas con palabras en mayúscula. Dicho esto, vamos a conocer y comprender algunos mecanismos gramaticales de la lengua de signos.

- Cuando varias glosas se unen para formar la descripción de un signo, no debe haber un espacio en blanco. Los espacios en blanco sirven para diferenciar las glosas correspondientes a signos diferentes.

- El carácter "+" significa que las dos glosas componen un signo compuesto.

  Ej: "*pares*" → *PAPA+MAMA*

- El carácter "-" significa que esas glosas, aunque en castellano/catalán correspondan con diferentes palabras, en LSC son un único signo.

  Ej: "*tot el dia*" → *TOT-EL-DIA*

- El verbo: Uno de los aspectos más importantes a tener en cuenta es que en la lengua de signos el verbo se incluyen normalmente al final de la frase, es decir, utilizando la sintaxis *Sujeto – Objeto – Verbo*. La oración comienza con la información más importante.

  Ej: "*Comprar una casa*" → *CASA COMPRAR*

- Sujeto: En castellano/catalán podemos eliminar el sujeto de las frases ya que el propio verbo nos da la información necesaria para saber quién realiza la acción; sin embargo, la sintaxis de la lengua de signos nos obliga a poner el sujeto en todo el momento.

  Ej: "*Menjo una pera*" → *PERA YO MENJAR*

- Interrogativas: En cuanto a las oraciones interrogativas normalmente las partículas interrogativas se incluyen al final de la frase elevando las cejas en el caso de preguntas cerradas (aquellas respuestas SÍ o NO) y frunciéndolas en las abiertas (las que contienen las partículas interrogativas qué, quién, cómo, dónde, cuál, por qué).

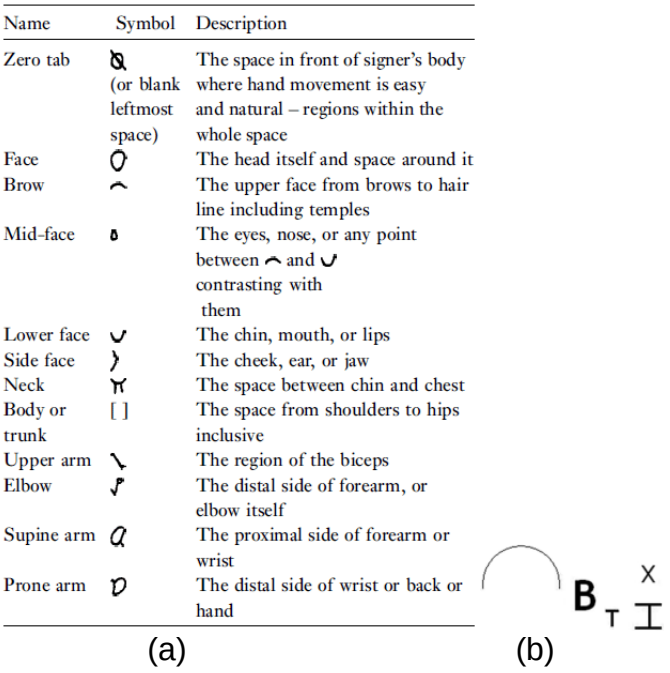
- Negativas: En relación a las oraciones negativas, al igual que ocurre en el caso de las interrogativas, las partículas negativas suelen incluirse al final de la frase, detrás del verbo, negando con la cabeza.

  Ej: "*No, no sóc Sorda*" → *YO SORDA* NO (negación con la cabeza).

1.4. Sistema de transcripción

La necesidad de representar por escrito la producción signada se remonta a los inicios de la investigación lingüística de las lenguas de signos. *William C. Stokoe* no sólo ha pasado a la historia por su pionero análisis lingüístico de La lengua de Signos Americana (ASL), sino también por el hecho de haber sido el primero en crear un sistema de notación que permitiera la transcripción de los elementos que constituían los signos de la ASL.

Sus parámetros engloban la configuración de la mano, el lugar de articulación y el movimiento, y su sistema significó la creación de símbolos con los que escribir los cincuenta y cinco "fonemas" que constituían estos tres parámetros formativos. Stokoe diseñó este sistema para que pudiera escribirse con una máquina de escribir empleando una fuente especial. Procuró utilizar símbolos que fueran tan icónicos como fuera posible con respecto de la localización y el movimiento.



Name	Symbol	Description
Zero tab	(or blank leftmost space)	The space in front of signer's body where hand movement is easy and natural – regions within the whole space
Face		The head itself and space around it
Brow		The upper face from brows to hair line including temples
Mid-face		The eyes, nose, or any point between and contrasting with them
Lower face		The chin, mouth, or lips
Side face		The cheek, ear, or jaw
Neck		The space between chin and chest
Body or trunk	[ ]	The space from shoulders to hips inclusive
Upper arm		The region of the biceps
Elbow		The distal side of forearm, or elbow itself
Supine arm		The proximal side of forearm or wrist
Prone arm		The distal side of wrist or back or hand

(a) (b)

Fig. 1.2 a) Notación Stokoe para describir el lugar de articulación de la mano.
b) Representación del signo "CONOCER"

Algunos inconvenientes de la notación Stokoe es que el sistema se basaba en la Lengua de Signos ASL, con lo cual no se reflejaba todos los movimientos posibles en otras lenguas de signos. Otro de los inconvenientes de este sistema de notación es que no se tienen en cuenta los componentes no manuales de los signos (boca, ojos, cejas, hombros, etc.).

Posteriormente y a medida que se ha ido avanzando en la investigación lingüística de las diferentes lenguas de signos, los investigadores se han visto en la necesidad de desarrollar diversos sistemas de notación que contemplaban o modificaban el sistema desarrollado por Stokoe. Uno de ellos es el que vamos a analizar a continuación y el que utilizaremos en nuestro proyecto para la transcripción de los signos.

1.4.1. HamNoSys

HamNoSys ó Sistema de Notación de Hamburgo es un sistema de transcripción "fonética" de las lenguas de signos. Su principal característica frente a otros sistemas de notación es que no se concibió para la transcripción de una lengua de signos en particular, sino como un intento de crear un sistema internacional de notación por ordenador para la investigación internacional de las lenguas de signos. Por esta razón, no se basa en un alfabeto dactilológico concreto, rompiendo así con la tradicción iniciada por Stokoe.

HamNoSys fue creado en el Center for German Sign Language and Comunicaction on the Deaf de la Universidad de Hamburgo alrededor de 1987 y consta de más de 150 símbolos. Incluye símbolos que representan las configuraciones de la mano, la orientación, el lugar y el movimiento. Los símbolos se disponen de una forma lineal y con un orden fijo.

Hasta ahora, HamNoSys se ha empleado como un sistema de notación de referencia en varios proyectos de investigación en lengua de signos. En primer lugar, es el sistema de notación más estable y más frecuentemente utilizado entre la comunidad de lengua de signos. Por último, las notaciones son más compactas y sencillas de introducir en un entorno de edición de signos.

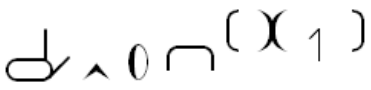


Fig. 1.3 Representación del signo "ALEMANIA" en HamNoSys

En la web de la Universidad de Hamburgo existe un tutorial que explica el significado de cada signo y los pasos a seguir para formar una palabra (ver [9]). En el anexo D hemos elaborado una lista completa de los símbolos HamNoSys.

1.4.2. SiGML

(Signing Gesture Markup Language) es un tipo de lenguaje del formato XML que permite la transcripción de los gestos de la lengua de signos. Dado que los ordenadores no pueden procesar la sintaxis de estas descripciones en HamNoSys, la

Universidad Est Anglian diseñó SIGML, una versión modificada de XML y que generó un traductor de HamNoSys a SIGML.

La descripción en SIGML de un signo contiene exactamente la misma información que la descripción en HamNoSys, pero en un lenguaje que pueden procesar los ordenadores.

HamNoSys	**SiGML**
	<sigml> <hns_sign gloss="Area"> <hamnosys_nonmanual> </hamnosys_nonmanual> <hamnosys_manual> <hamfinger2345/> <hampinky/> <hamextfingerol/> <hampalmd/> <hamchest/> <hamcircleu/> <hamrepeatfromstart/> </hamnosys_manual> </hns_sign> </sigml>

Tabla 1.2 Codificación en SiGML

Cada signo se diseña dentro de la etiqueta <hns_sign> y en el que está compuesto por dos etiquetas internas (<hamnosys_nonmanual> y <hamnosys_manual>. Dentro de la etiqueta <hamnosys_nonmanual> se define los movimientos de boca, cabeza, nariz, ojos, cejas y hombros. En la etiqueta <hamnosys_manual> se describe los movimientos de las manos.

La notación SiGML se ha desarrollado en EUA para apoyar el trabajo de los proyectos ViSiCAST y eSIGN (ver [3] y [4]).

CAPÍTULO 2. LA TRADUCCIÓN AUTOMÁTICA

2.1. Introducción a la traducción automática

La traducción automática (TA) es una rama de la Lingüística Computacional y accesible desde muchos puntos de vista (informático, lingüístico, empresarial, etc).

A principios de los años 50 y comienzos de los 60 del siglo XX existía entre algunos técnicos en inteligencia artificial estadounidenses la confianza de que la tarea de la traducción se podría automatizar, y que existirían sistemas capaces de traducir cualquier texto. Dado que las máquinas son más baratas de mantener que los traductores humanos y además pueden producir mucho más y en menos tiempo, la TA se perfilaba como una línea de investigación que podía ser aplicada para reducir los costes de traducción de las empresas, los organismos internacionales y los servicios de inteligencia militar.

Los sistemas de TA permiten traducir amplios cuerpos de texto en un tiempo inferior a la traducción humana. Proyectos como la edición de la versión en catalán de “El Periódico” no serían posibles si no se llevaran a cabo con un sistema de TA. Por otra parte, para organismos internacionales como la Comunidad Europea, que tiene que generar grandes volúmenes de documentos en muchas lenguas en un tiempo limitadamente corto, la TA se ha convertido también en una necesidad. Por esta razón la Comunidad financió el proyecto Eurotra, que consistió en la preparación de un sistema capaz de traducir automáticamente su documentación en las lenguas oficiales de la Unión Europea.

La TA disminuye costes cuando se trata de traducir constantemente documentos escritos en un lenguaje controlado. Un documento está escrito en un lenguaje controlado si tiene unas estructuras sintácticas simples y rígidas, no es ambiguo, su léxico es limitado y tiene una fraseología formada previamente.

2.1.1. Limitaciones

Las restricciones de un sistema de TA conmueven sobre todo a la calidad de la traducción. Si un sistema de TA no tiene una representación adecuada del significado de la frase original es más que probable que la traducción no se entienda o sea ilógica.

La compresión de una frase requiere de un conocimiento complejo de la lengua origen, de unos elementos que procesen la información lingüística y de conocimiento del mundo contenida en la frase. Evidentemente, el procesamiento de todo ella tendría un enorme coste en tiempo y probablemente los recursos de memoria del sistema se colapsarían precipitadamente.

Ahora bien, la precisión en la designación de conceptos puede mejorar mediante la consulta automática a bases de datos terminológicas de un dominio concreto en el par de lenguas del sistema. No todos los sistemas de TA permiten que los usuarios incorporen base de datos terminológicos.

2.1.2. Tipos de Traducción Automática

Se pueden diferenciar dos tipos principales de sistemas de traducción automática: los que se basan en reglas lingüísticas y los que se basan en corpus textuales.

- **Basados en reglas lingüísticas:** Consisten en sustituir las palabras por su equivalente más cercano. Primero se hace una representación simbólica interna del texto original. Desde ahí se puede hacer la traducción palabra por palabra o utilizando una intermedia.

- Por transferencia: Se analiza el original que da paso a una representación interna que será el enlace a otros idiomas.

- Por lenguaje intermedio: El texto base se convierte en un lenguaje intermedio con estructura diferente al lenguaje origen y al lenguaje destino.

- **Basados en corpus textuales:** Se basa en un corpus lingüístico que se ha obtenido de muestras reales.

- Estadística: Se obtienen corpus de textos bilingües como fuente. Actualmente la investigación en traducción automática se ha centrado en estos sistemas porque los resultados obtenidos, sobre todo cuando se trata de lenguas cercanas, son bastantes prometedores y los costes en tiempo y dinero de su construcción son menores que para la creación de motores de traducción con conocimiento lingüístico. Consiste en buscar las palabras de la lengua destino que traducen mejor las palabras de la oración original y en encontrar la secuencia de estas palabras que es más adecuada para ser una oración correcta en la lengua destino. Los cálculos de las probabilidades son significativos si los corpus son muy grandes.

- Basado en ejemplos: Parte de un corpus bilingüe como fuente. Se basa en analogías. Resuelve un problema tomando como base otras soluciones similares ya resueltas.

- Basado en el contexto: Consiste en traducir cada palabra teniendo en cuenta las palabras que le rodean. Divide el texto en unidades de cuatro a ocho palabras y las traduce al idioma destino. Después se eliminan aquellas que han generado frases sin sentido. Luego se mueve la ventana una posición o palabra volviendo a traducir de nuevo dejando solo las frases con sentido. Se repite este proceso de nuevo

hasta terminar todo el texto. Después se unen los resultados de cada ventana de modo que quede un texto unitario.

	Ventajas	**Inconvenientes**
Basados en reglas lingüísticas	Buenos resultados en las traducciones	Más complejo, alto coste en inversión de capital humano
Basados en corpus textuales	Bajo coste de tiempo y dinero	Resultados pobres si no disponemos de un gran corpus paralelo

Tabla 2.1 Ventajas e inconvenientes de los diferentes tipos de traducción automática

2.2. SMT Moses

SMT Moses (Statistical Machine Translation Moses) es un sistema de Traducción Automática Estadístico de código abierto, bajo licencia LGPL. El desarrollo de Moses se apoya principalmente en los proyectos EuroMatrix y EuroMatrixPlus, financiados por la Comisión Europea y recibe apoyo adicional de: Universidad de Edimburgo (Escocia), RWTH Aachen (Alemania), Fundación Bruno Kessler de Trento (Italia), Universidad de Maryland (EE.UU), Instituto Tecnológico de Massachusetts (EE.UU), Universidad Charles (Praga), las agencias de financiamiento de DARPA, NSF, el Departamento de Defensa EE.UU y financiación de la UE a través del proyecto TC-Star.

El sistema Moses se ejecuta en Linux. Es posible ejecutarlo desde Windows pero utilizando sobre Cygwin, que emula la ejecución de Unix en Windows. Las secuencias de comandos de entrenamiento y puesta a punto fueron desarrolladas para Unix como sistema operativo por lo que es difícil de ejecutar sin el entorno Cygwin. Además, la instalación de Cygwin requiere algunos cambios en los Makefiles y scripts debido a diferencias entre Cygwin y Linux. Para evitar problemas, en este proyecto instalaremos el sistema Moses en una máquina con Linux.

En el anexo A hemos elaborado una guía de instalación del sistema Moses.

2.2.1. Funcionamiento

Moses permite entrenar de forma automática los modelos de traducción para cualquier par de lenguas. Todo lo que necesitas es una colección de textos traducidos (corpus paralelo). Un algoritmo de búsqueda eficaz encuentra rápidamente la traducción, aplicando el Teorema de Bayes calcula la probabilidad de que la cadena del idioma destino (d) haya sido generada por la cadena origen (o). De tal manera que para conseguir p(d|o) se calcula p(o|d) · p(d), donde p(o|d) es la probabilidad de que la cadena origen sea la traducción de la cadena destino (modelo de traducción), y p(d) es

la probabilidad de ver aquella cadena destino (modelo de lenguaje de la lengua de destino). Matemáticamente, encontrar la mejor traducción õ se consigue escogiendo aquella secuencia de signos que permita obtener la probabilidad máxima.

$$\tilde{O} = \arg\max p\,(d|o) = \arg\max p(o|d) \cdot p(d) \qquad \textbf{(2.1)}$$

Modelo de lenguaje

Es un mecanismo para definir la estructura del lenguaje, es decir, para restringuir adecuadamente las secuencias de unidades lingüísticas más probables. En general son útiles en aplicaciones que exhiban una sintaxis y/o semántica compleja. Un buen modelo de lenguaje solamente debería aceptar (con alta probabilidad) frases correctas y rechazar (o asignar baja probabilidad a) aquellas secuencias de palabras incorrectas. Para generar el modelo de lenguaje se emplea la herramienta SRILM (SRI Languages Modeling Toolkit), que permite estimar modelos de lenguaje tipo n-grama. Un modelo de n-grama determina la probabilidad de una palabra dadas las n-1 palabras previas.

Modelo de traducción

Para generar el modelo de traducción se emplea la herramienta GIZA++. Además, se necesita una colección de textos en lengua origen traducidos a lengua destino (corpus paralelo).

GIZA++ es una herramienta software que permite alinear textos y aprender modelos de traducción basada en palabras a partir de ellos, empleando en este sistema 5 iteraciones. De tal manera que, a partir del corpus bilingüe, se obtiene el alineamiento en los dos sentidos: origen-destino y destino-origen, para posteriormente combinar dichos alineamientos. Esta combinación puede realizarse de varias formas que se comentan a continuación.

- **Destino-Origen (DO):** Sólo se tiene en cuenta el alineamiento en un sentido, ignorando el alineamiento en sentido contrario.

- **Origen-Destino (OD):** En este caso el alineamiento que se tiene en cuenta es el contrario.

- **Intersección (I):** Partiendo de los dos alineamientos anteriores, se obtiene como alineamiento final los alineamientos intersección de ambos. Este tipo de alineamiento es el más exigente: permite tener alineamientos más fiables pero a costa de tener menos. Esta característica influye en la calidad del modelo de traducción cuando no se dispone de una gran cantidad de frases para su entrenamiento.

- **Unión (U):** En este caso, se toma la unión de los puntos de alineamiento en los dos sentidos. De esta manera, se obtiene puntos adicionales de alineamiento, consiguiendo más ejemplos para entrenar el modelo de traducción de palabras, pero también la calidad de los alineamientos considerados es menor que en el caso de la intersección.

- **Crecimiento (C):** En este tipo de alineamiento se toman los puntos de alineamiento de la intersección y, a continuación, se añaden los puntos de la unión que estén contiguos a los puntos de la intersección. Con esta probabilidad, se intenta buscar una situación intermedia entre la unión y la intersección consideradas anteriormente. El objetivo es buscar un punto de equilibrio entre cantidad de puntos de alineamiento y su calidad.

- **Crecimiento diagonal (CD):** Igual que el alineamiento anterior, pero añadiendo sólo los puntos de la unión contiguos a la intersección y situados en la diagonal.

- **Crecimiento diagonal evitando casos de no alineamiento (CDP):** Este tipo de combinación es similar a la anterior con un postproceso adicional que tiene como objetivo evitar que ninguna palabra o signo se quede sin ningún punto de alineamiento. En el caso de que eso ocurra se añaden los alineamientos en uno u otro sentido necesarios.

A partir de dicho alineamiento, se obtienen las probabilidades de traducción para todos los pares de palabra (w(d|o) y w(o|d)), realizándose así una estimación de la tabla de traducción léxica más probable. A continuación, de todos los pares de subsecuencias obtenidos, se escogen con el programa phrase_extract sólo los que sean consistentes con el alineamiento de palabras. Y, por último, con el programa phrase_score, se calculan las probabilidades de traducción para todos los pares de subfrases en los dos sentidos. Estos dos últimos programas (phrase_extract y phrase_score) ya están incluidos dentro de la herramienta GIZA++.

Los modelos de traducción y de lenguaje (en lengua destino) se combinan linealmente para ser utilizados como heurístico en el proceso de búsqueda necesario para generar la traducción de una frase dada. Para ello, con el traductor Moses se traduce y evalúa una lista de frases de validación cuya traducción correcta se conoce, probando con distintos pesos aleatoriamente y escogiendo los que dan los mejores resultados.

Por último, para la traducción se emplea el decodificador Moses, que implementa un algoritmo de búsqueda para obtener, a partir de una frase de entrada, la secuencia de palabras que con mayor probabilidad corresponde a su traducción. Para ello, utiliza los modelos de traducción y lenguaje obtenidos anteriormente, con los pesos de sus probabilidades ajustados.

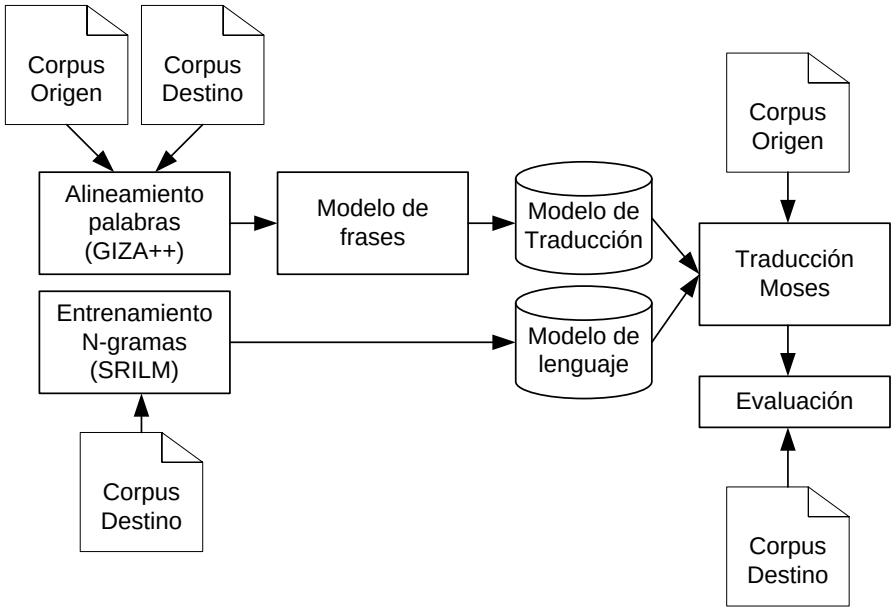


Fig. 2.1 Arquitectura del Sistema Moses

2.2.2. Preparar corpus paralelo

Los datos del corpus paralelo han de cumplir los siguientes puntos:

- Una frase por línea, no líneas vacías.
- Las frases con más de 100 palabras (y su correspondiente traducción) tienen que ser eliminadas (tenga en cuenta que un límite de longitud menor de la frase acelerará el entrenamiento).
- Todo en minúscula.
- Las palabras han de estar separadas de los signos de puntuación y ortográficos.

2.2.3. Crear modelo de lenguaje

Para crear el modelo de lenguaje utilizamos el programa *ngram-count* de la herramienta SRILM. Su sintaxis es la siguiente:

```
./srilm/bin/i686/ngram-count -order 3 -interpolate -kndiscount -text
meteo.lsc -lm meteo.lm
```

-order: Establecer el orden de máxima (longitud) de N-gramas para contar. El orden por defecto es de 3.
-text: Se indica la ruta del corpus de la lengua destino.
-lm: Se define un archivo para guardar el modelo de lenguaje.

Desde la web http://www-speech.sri.com/projects/srilm/manpages/ngram-count.1.html explica con detalle cada uno los parámetros que podemos incluir.

2.2.4. Entrenamiento

Por último, entrenamos el corpus paralelo utilizando el script *train-model.perl*. Su sintaxis es la siguiente:

```
./moses-scripts/scripts-YYYYMMDD-HHMM/training/train-model.perl -
scripts-root-dir /moses-scripts/scripts-YYYYMMDD-HHMM/ -root-dir
/home/javi.marin/demo/work -corpus
/home/javi.marin/demo/work/corpus/meteo -f cat -e lsc -alignment grow-
diag-final-and -reordering msd-bidirectional-fe -lm
0:3:/home/javi.marin/demo/work/lm/meteo.lm >& work/training.out &
```

-scripts-root-dir: ruta de directorio de la carpeta scripts-YYYYMMDD-HHMM/.
-root-dir: ruta de directorio raíz del corpus paralelo.
-corpus: ruta de directorio de la carpeta donde se encuentra el corpus paralelo.
-f: nombre de extensión del archivo de la lengua origen (“cat”).
-e: nombre de extensión del archivo de la lengua destino (“lsc”).
-lm*:* ruta de archivo del modelo de lenguaje. El “0” indica el número de modelos de lenguajes utilizados empezando por el 0. El “3” indica el número de n-gramas utilizado en ese modelo de lenguaje.

El entrenamiento habrá sido un éxito si en la última línea del archivo training.out indica que se ha creado el archivo de configuración “moses.ini”.

Este archivo de configuración contiene todas las rutas para el modelo generado y una serie de ajustes de los parámetros por defecto. Más adelante veremos cómo usar este archivo en la etapa de traducción.

2.2.5. Tuning

El script de entrenamiento *train-model.perl* genera un archivo de configuración “moses.ini” con unos pesos por defecto de baja calidad. Es por eso que necesitamos obtener mejores pesos mediante la optimización de rendimiento de la traducción del modelo generado.

En la traducción automática estadística, los modelos probabilísticos son usados para encontrar la mejor traducción *e** de un determinada frase de origen *f*, entre todas las traducciones posibles *e*. La búsqueda de la mejor traducción se conoce como descodificación. Los modelos probabilísticos son estimados desde los datos de entrenamiento, y puede incluir modelos de traducción, modelos de lenguajes, modelos de reordenamiento, etc. Con el fin de combinar las pruebas de diferentes modelos, es una práctica habitual de utilizar un modelo lineal, con las probabilidades de registro

como características. Si las características de los modelos son h_1, …,h_r, que depende de *e* i *f*, entonces la mejor traducción se da por:

$$e^*(\lambda) = \arg\max \sum_{i=1}^{r} \lambda_i h_i(e, f) \quad \textbf{(2.2)}$$

Esto se hace con el script de tuning *mert-moses.pl* que mejora la tasa de error mínimo del entrenamiento, es decir, encuentra un juego de pesos que ofrecerá la mejor calidad de traducción. Internamente, el algoritmo del *mert-moses.pl* consiste en un bucle externo y otro bucle interno (ver figura 2.2). El bucle externo ejecuta el decodificador Moses sobre la frase de origen con un conjunto de pesos por defecto, generando listas de n-best de traducciones, finalmente llama al bucle interno para optimizar los pesos basándose en la lista de n-best, repitiendo el proceso hasta que los nuevos pesos ya no se cambian.

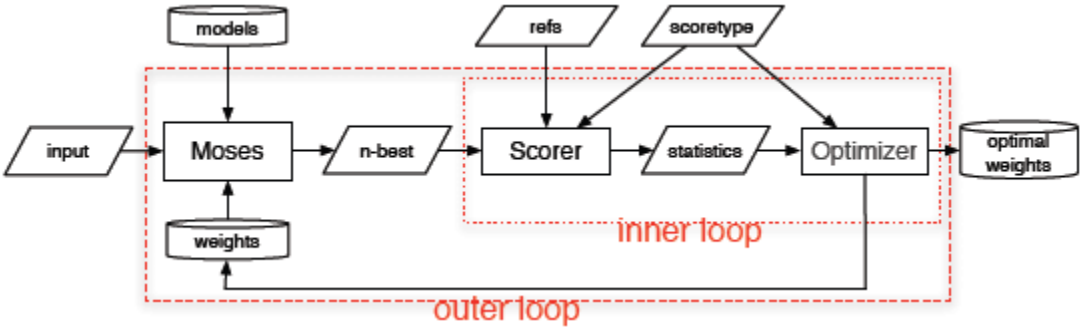


Fig 2.2 Bucle externo e interno de MERT

La sintaxis para ejecutar el script *mert-moses.pl* es la siguiente:

```
./moses-scripts/scripts-YYYYMMDD-HHMM/training/mert-moses.pl
/work/corpus/meteo.cat        work/corpus/meteo.lsc        /moses/moses-
cmd/src/moses /work/model/moses.ini -working-dir /work/tuning/mert/
-decoder-flags "-drop-unknown -input-factors 0 -output-factors 0"
```

Se creará un nuevo archivo de configuración “moses.ini” dentro del directorio /work/tuning/mert/ con los pesos optimizados. Insertaremos sus pesos en el archivo de configuración inicial generado por la etapa de entrenamiento utilizando el script reuse-*weights.perl* de la siguiente manera:

```
/scripts/reuse-weights.perl        /work/tuning/mert/moses.ini        <
/work/model/moses.ini > /work/tuning/moses-tuned.ini
```

Esto generará el archivo de configuración final “moses-tuned.ini” con los pesos optimizados de la etapa de tuning.

2.2.6. Traducción

Este apartado describe el uso del decodificador del sistema Moses. Previamente se supone que ya tenemos el modelo de entrenamiento creado en las etapas de entrenamiento y tuning. Sin más preámbulos, vamos a ejecutar el decodificador:

```
echo 'dilluns , 19 de novembre de 2007' | moses/moses-cmd/src/moses -f
/work/model/moses.ini > out.txt

Defined parameters (per moses.ini or switch):
      config: /home/javi/Escritorio/work/model/moses.ini
      distortion-file:     0-0     wbe-msd-bidirectional-fe-allff     6
/home/javi/Escritorio/work/model/reordering-table.wbe-msd-
bidirectional-fe.gz
      distortion-limit: 6
      input-factors: 0
      lmodel-file: 0 0 3 /home/javi/Escritorio/work/meteo.lm
      mapping: 0 T 0
      ttable-file:  0  0  0  5  /home/javi/Escritorio/work/model/phrase-
table.gz
      ttable-limit: 20
      weight-d: 0.3 0.3 0.3 0.3 0.3 0.3 0.3
      weight-l: 0.5000
      weight-t: 0.2 0.2 0.2 0.2 0.2
      weight-w: -1
Loading lexical distortion models...have 1 models
Creating lexical reordering...
weights: 0.300 0.300 0.300 0.300 0.300 0.300
Loading table into memory...done.
Start  loading  LanguageModel  /home/javi/Escritorio/work/meteo.lm  :
[0.000] seconds
Finished loading LanguageModels : [0.000] seconds
Start  loading  PhraseTable  /home/javi/Escritorio/work/model/phrase-
table.gz : [0.000] seconds
filePath: /home/javi/Escritorio/work/model/phrase-table.gz
Finished loading phrase tables : [0.000] seconds
IO from STDOUT/STDIN
Created input-output object : [0.000] seconds
Translating: dilluns , 19 de novembre de 2007

Collecting options took 0.000 seconds
Search took 0.030 seconds
BEST TRANSLATION: després dilluns dia 19 mes novembre any 2007
[1111111]  [total=1.607] <<0.000, -8.000, 0.000, -0.437, 0.000, 0.000,
-0.336, 0.000, 0.000, -7.925, 0.000, -3.116, -0.405, -9.471, 2.000>>
Translation took 0.030 seconds
Finished translating
```

Para ejecutar el decodificador es necesario pasar los parámetros de la frase de entrada ("`'dilluns , 19 de novembre de 2007`") y la ruta del archivo de configuración

moses.ini (“`/work/model/moses.in`”). Finalmente podemos guardar el resultado en un nuevo archivo (“> out.txt”). La mejor traducción encontrada por el decodificador Moses ha sido: “`després dilluns dia 19 mes novembre any 2007`”.

2.2.7. Evaluación

Con el objetivo de evaluar la calidad de la traducción, es necesario comparar la salida del sistema Moses con una referencia y calcular algunas medidas de evaluación.

BLEU (Bilingual Evaluation Understudy) es un método de evaluación de la calidad de traducciones realizadas por sistemas de traducción automática. Una traducción tiene mayor calidad cuanto más similar es con respecto a otra de referencia, que se supone correcta. Puede calcularse utilizando más de una traducción de referencia. Esto permite una mayor robustez a la medida frente a traducciones libres realizadas por humanos. Se calcula normalmente a nivel de frases y halla la precisión en n-gramas entre la traducción del sistema y la de referencia. Compara los n-gramas de la frase generada por el sistema de traducción con los n-gramas de la frase de referencia, contando el número de n-gramas que coinciden independientemente de la posición.

Otra medida es NIST, que se basa en la BLEU con algunas modificaciones: el primer lugar, BLEU utiliza la media geométrica de la precisión de los N-gramas, mientras que NIST utiliza una media aritmética para reducir el impacto de bajas concurrencias para órdenes altos de n-gramas. También, BLEU calcula la precisión de n-gramas utilizando pesos iguales para cada n-grama, mientras que NIST considera la calidad de la información que proporciona un n-grama particular en sí mismo (por ejemplo, cuanto menos frecuente sea un n-grama más peso se le asignará).
Para ello, vamos a utilizar la herramienta de NIST *mteval-v12b.pl* que evaluará la calidad de la traducción. Su sintaxis es la siguiente:

```
/tools/mteval-v12b.pl -s /work/corpus/meteo.cat.sgm -r
/work/corpus/meteo.lsc.sgm -t /work/corpus/out.sgm -c
```

Donde “meteo.cat.sgm”, “meteo.lsc.sgm” y “out.sgm” son respectivamente los archivos de origen, de referencia (destino) y de test (la traducción del archivo de origen con el sistema Moses).

La salida de este script muestra los valores de evaluación del sistema NIST y BLEU. El rango de valores para el sistema BLEU están entre 0 y 1, mientras el valor máximo de NIST depende del tamaño del conjunto de datos. En los dos casos, cuanto mayor es el valor mejor es la traducción.

Estos tres archivos deben estar en un formato de tipo XML para ser evaluados correctamente. En el anexo B podemos ver la estructura de estos archivos.

CAPÍTULO 3. DISEÑO DEL SISTEMA

3.1. Arquitectura cliente/servidor

En el mundo de TCP/IP las comunicaciones entre computadoras se rigen básicamente por lo que se llama modelo Cliente/Servidor, éste es un modelo que intenta proveer usabilidad, flexibilidad, interoperabilidad y escalabilidad en las comunicaciones.

Es una arquitectura distribuida que permite a los usuarios finales obtener acceso a la información en forma transparente aún en entornos multiplataforma. El cliente envía un mensaje solicitando determinado servicio a un servidor (hace una petición), y este envía uno o varios mensajes con la respuesta (provee el servicio).

Esta arquitectura es una extensión de programación modular en la que la base fundamental es separar una gran pieza de software en módulos con el fin de hacer más fácil el desarrollo y mejorar su mantenimiento.

En definitiva, un sistema Cliente/Servidor es un sistema de información distribuido basado en las siguientes características:

- Servicio: El servidor los proporciona y el cliente los utiliza.
- Recursos compartidos: Muchos clientes utilizan los mismos servidores y, a través de ellos, comparten tanto recursos lógicos como físicos.
- Protocolos asimétricos: Los clientes inician “conversaciones”. Los servidores esperan su establecimiento pasivamente.
- Independencia de la plataforma hardware y software que se emplee.
- Sistemas débilmente acoplados. Interacción basada en envío de mensajes.
- Escalabilidad horizontal (añadir clientes) y vertical (ampliar potencia de los servidores).
- Integridad: Datos y programas centralizados en servidores facilitan su integridad y mantenimiento.

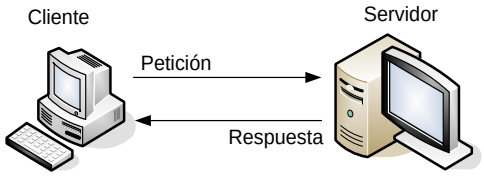


Fig. 3.1 Modelo Cliente/Servidor

Para el presente proyecto, la idea es que un usuario pueda descargar desde un servidor Web las aplicaciones Cliente y Editor. La aplicación Cliente conectará con una aplicación Servidor para recibir la traducción en la lengua de signos a partir de un texto que el cliente envíe. De manera que la aplicación Servidor debe tener, además del acceso al sistema Moses, un servidor de base de datos para recoger la codificación SiGML de la secuencia de signos que Moses ha obtenido. Por último, la aplicación Cliente enviará la codificación SiGML obtenida al avatar.

En cuanto a la aplicación Editor, también conectará con el servidor de base de datos para poder extraer y editar un signo.

Así pues, el esquema final del proyecto que presentaremos se muestra a continuación (Fig. 3.2), utilizando sockets de red (canales de comunicación) entre las aplicaciones y el avatar para enviar o recibir datos.

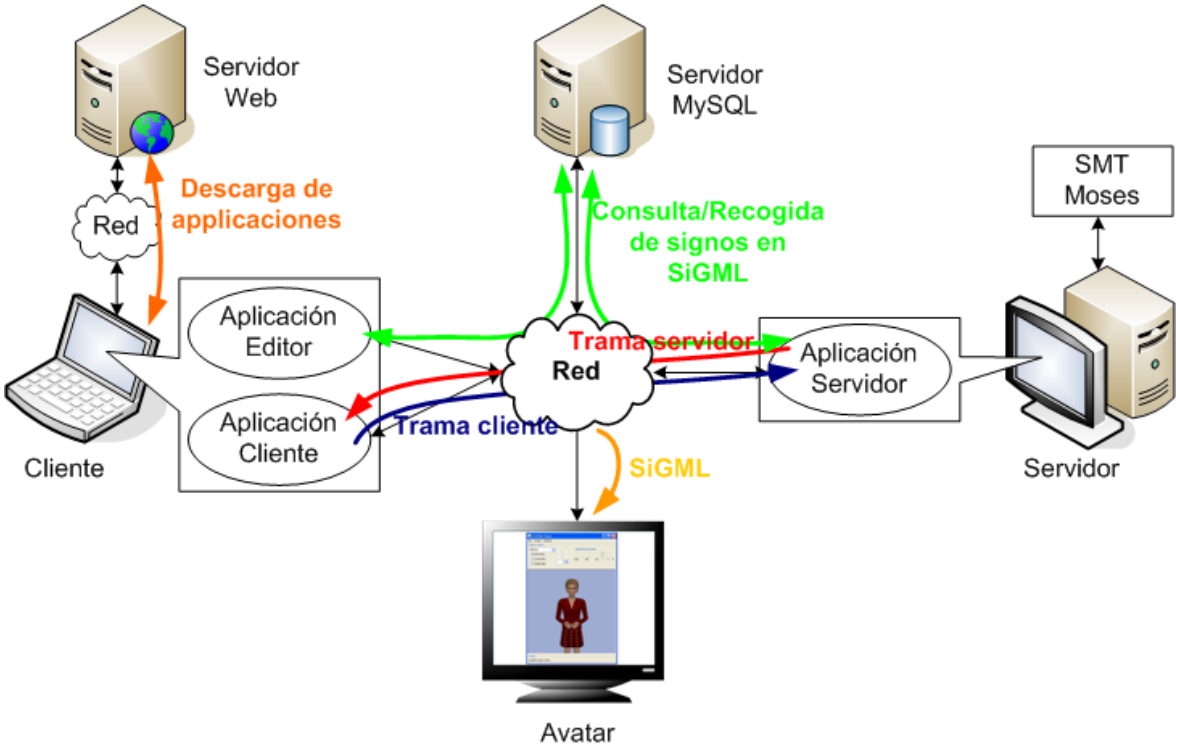


Fig. 3.2 Esquema final del proyecto

3.2. Herramientas de desarrollo

3.2.1. Java SE

Como lenguaje de programación, podemos decir que Java es un lenguaje de alto nivel, orientado a objetos, robusto, seguro… pero si alguna característica destaca en especial es que las aplicaciones creadas con Java son independientes del hardware en el que se ejecutan, lo que permite “programar una vez y ejecutar en muchos sitios”.

Esta característica es la que hace a Java tan apropiado para aplicaciones corporativas y de internet, donde es fácil pensar que conviven distintas plataformas hardware: Windows, Linux, Unix, Mac, etc. Al compilar un programa escrito en Java, se genera código independiente de la plataforma en la que se ha creado. Este código se conoce como bytecode y es interpretado en el ordenador en el que realmente se ejecuta.

Para que esto sea posible, es necesario que el ordenador que ejecuta el bytecode sepa interpretarlo. Y para ello, debe disponer de lo que se conoce como una máquina virtual de Java (JVM). La máquina virtual de Java, por defecto, no viene instalada en los ordenadores, por lo que debemos obtener e instalarlo de alguna forma, como puede ser desde el sitio web de Sun.

Sun Microsystem dividió Java en tres grandes ramas, atendiendo al mercado a que va dirigido, cada una de ellas con su conjunto de APIs y herramientas de desarrollo propias: grandes ordenadores, ordenadores de sobremesa y microordenadores o dispositivos de memoria limitada.

Java SE es la edición que va orientada a ordenadores de sobremesa y el que usaremos sus clases para el desarrollo de nuestras aplicaciones. Comprende el JDK (Kit de Desarrollo de Java) hasta ahora distribuido por Sun, en donde Swing se ha convertido en pieza clave y al que se han incorporado clases adicionales para facilitar el desarrollo de aplicaciones Java en donde la interfaz de usuario tiene una importancia muy especial. Ésta es la versión en la que la mayoría de la gente piensa cuando piensa en Java.

3.2.2. Entorno de desarrollo NetBeans

Una de las ventajas de la tecnología Java es que está estrechamente relacionada con el mundo “Open Source” o de código abierto, por lo que es fácil encontrar entornos de desarrollo o IDEs gratuitos y de gran calidad. Uno de ellos es el que patrocina Sun Microsystems, cuyo nombre es NetBeans y es el entorno de desarrollo que utilizaremos para la creación de las aplicaciones.

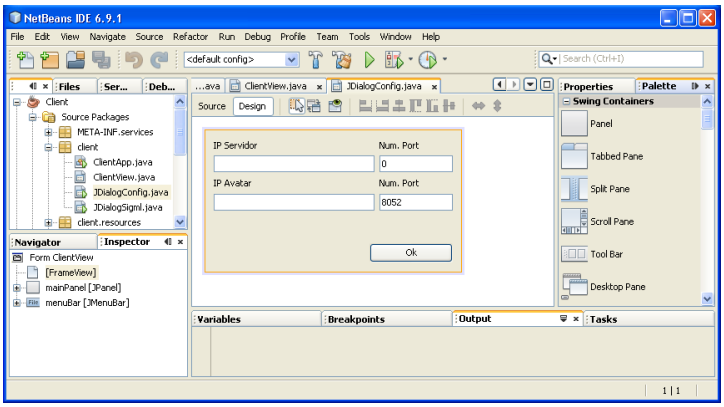


Fig. 3.3 NetBeans

3.2.3. Java Web Start

Java Web Start es una solución de distribución de aplicaciones basada en tecnología Java. Es la canalización entre Internet y el sistema que permite al usuario ejecutar y gestionar aplicaciones desde la Web.

Proporciona una activación fácil y rápida de las aplicaciones, garantiza la ejecución de la última versión de la aplicación, eliminando los complicados procesos de instalación o de modernización.

Cuando se descarga por primera vez una aplicación que utiliza la tecnología Java Web Start se ejecuta automáticamente y guarda la aplicación localmente, en la memoria del caché del equipo. De este modo, las subsiguientes ejecuciones son prácticamente instantáneas, ya que los recursos necesarios están disponibles de forma local. Cada vez que se inicia la aplicación, el componente de software de Java Web Start comprueba si en la sede Web de la aplicación hay una nueva versión disponible; si es así, la descarga y la ejecuta de forma automática.

Ejecutar una aplicación Java desde la web únicamente se requiere crear un archivo JNLP (Java Networking Launching Protocol). Cualquier enlace JNLP, al iniciar el proceso de ejecución, pide autorización al usuario. Además, las aplicaciones pueden estar firmadas para asegurar el remitente de la aplicación de modo que pueden seguir el modelo de seguridad de la plataforma Java 2 para asegurar la integridad de los datos que obtenemos a través de la red.

En el anexo C tenemos un ejemplo del formato de archivo .jnlp

3.2.4. XAMPP

Nuestro propósito es que el cliente pueda descargar y ejecutar las aplicaciones Cliente y Editor desde una página web. Para eso es necesario tener instalado un servidor web. Además, tanto las aplicaciones Servidor y Editor, necesitarán una base de datos para consultar signos y recoger su codificación SiGML.

XAMPP es una distribución multiplataforma libre que integra el servidor web Apache, la base de datos MySQL y los intérpretes para lenguajes script: PHP y Perl. Una vez instalado, accedemos a su panel de control y arrancamos el servidor web Apache y MySQL.

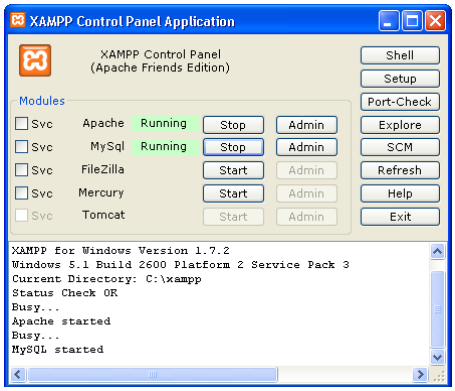


Fig. 3.4 Panel de control XAMPP

Para comprobar que funciona, abrimos nuestro navegador web y escribimos http://localhost. XAMPP mostrará su página inicial.

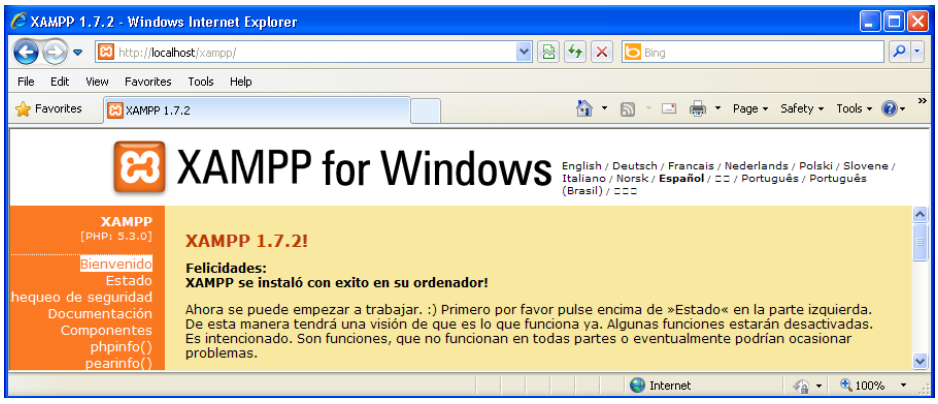


Fig. 3.5. Página inicial de XAMPP

El directorio de trabajo se encuentra ubicado en la subcarpeta “htdocs” del directorio XAMPP. Desde ahí, crearemos una carpeta y guardaremos los archivos JNLP y JAR de nuestras aplicaciones.

PhpMyAdmin

XAMPP permite configurar la base de datos MySQL utilizando la herramienta PhpMyAdmin (http://localhost/phpmyadmin/). Nos permite, entre otras opciones, crear, editar y eliminar base de datos. Desde ahí crearemos nuestra base de datos de signos, insertando su transcripción en glosa y su correspondiente codificación SiGML de todos sus movimientos. Desde Barcelona Media hemos podido almacenar un total de 260 signos aproximadamente.

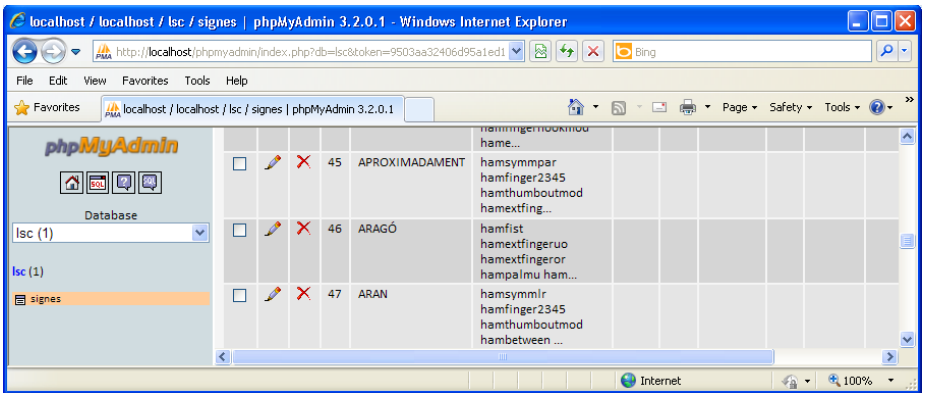


Fig. 3.6 Herramienta phpMyAdmin de XAMPP

3.2.5. SiGML Service Player

Este avatar sustituye al anterior software SiGMLSigning que fue desarrollado en Europa para los proyecto proyectos eSIGN y ViSiCAST y que proporcionaba la animación de secuencias de signos defindos en SiGML.

Esta última versión del avatar JASigning se puede ejecutar en Windows (XP, Vista, 7) y en las últimas versiones del MAC OS X 10.5 y 10.6. JASigning no es compatible con Linux en la actualidad.

Es preferible ejecutar JASigning con las nuevas actualizaciones de Java 6 (JRE), aunque debe funcionar con las últimas versiones de Java 5. Para los usuarios del Windows Xp, requiere la previa instalación del paquete Microsoft Visual C++ 2008 SP1 Redistributable Package, que lo podemos obtener desde la web oficial de Microsoft.

Desde la web Virtual Humans podemos descargar la aplicación JNLP SiGML Service Player. Esta aplicación reproduce una secuencia SiGML entregado en un socket de

red. Esta aplicación acepta solicitudes de conexión en el estándar TCP/IP con el número de puerto 8052.

3.3. Diagrama de caso de uso

En este diagrama identificamos previamente los actores del sistema y el listado de las acciones que pueden realizar. A continuación vamos a mostrar las acciones que aparecen en nuestro sistema para cada una de las aplicaciones que crearemos.

3.3.1. Diagrama de caso de uso en la aplicación Servidor

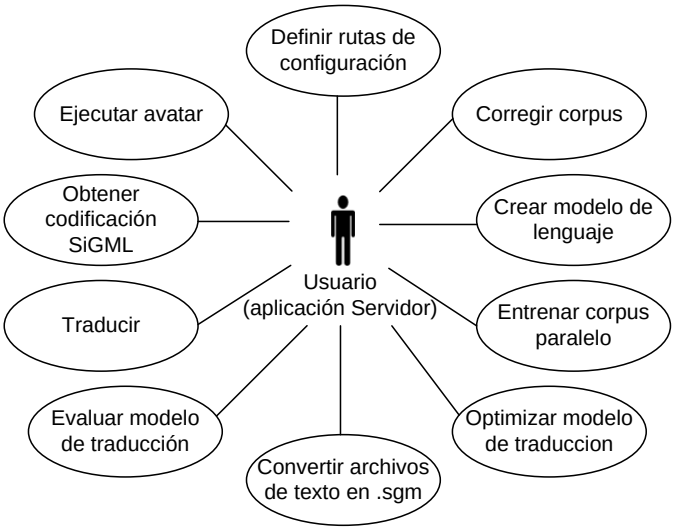


Fig. 3.7 Diagrama de caso de uso en la aplicación Servidor

- **Definir rutas de configuración:** Establece las rutas de las distintas herramientas y scripts que utilizará el sistema Moses, así como los datos de configuración del servidor MySQL y del Avatar.

- **Corregir corpus:** Corrige el corpus adaptándolo al formato requerido por Moses (convertir en minúscula, separar signos de puntuación, eliminar signos no ortográficos, etc.).

- **Crear modelo de lenguaje:** Ejecuta la herramienta SRILM del sistema Moses a partir del corpus de la lengua destino.

- **Entrenar corpus paralelo:** Ejecuta el script train-model.perl del sistema Moses para obtener un modelo de traducción.

- **Optimizar el modelo de traducción:** Ejecuta el script mert-moses.pl del sistema Moses para mejorar la tasa de error mínimo del entrenamiento.
- **Evaluar el modelo de traducción:** Ejecuta el script mteval-v12.pl para evaluar el modelo de traducción con los sistemas NIST y BLEU.

- **Convertir archivos de textos en forma .sgm:** Permite convertir los textos que serán evaluados en un formato que pueda ser procesado por los sistemas de evaluación NIST y BLEU.

- **Traducir texto:** Ejecuta el decodificador Moses para obtener la traducción de un texto de entrada. Permite activar la opción “Mostrar palabras desconocidas” en caso de que Moses no encontrara su traducción y la opción “Deletrear” para aquellos signos, obtenidos por Moses, que no se encuentren en la base de datos y quisiéramos deletrearlos. Útil para traducir nombres propios donde no existe un signo específico (ej: “JAVIER”).

- **Obtener codificación SiGML:** Establece una conexión con el servidor MySQL para consultar y recoger la codificación SiGML de la secuencia de signos obtenidos por el decodificador Moses.

- **Ejecutar Avatar:** Envía la codificación SiGML obtenida al Avatar.

3.3.2. Diagrama de caso de uso en la aplicación Cliente

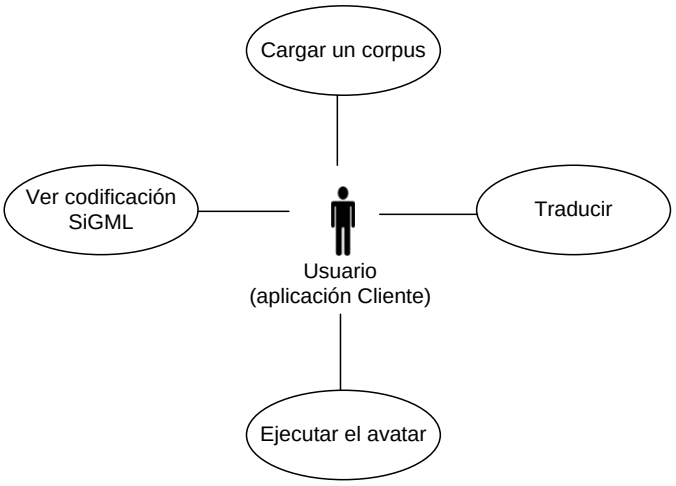


Fig. 3.8 Diagrama de caso de uso en la aplicación Cliente

- **Cargar un corpus:** Permite seleccionar un corpus para mostrar las frases por pantalla y seleccionar la que queramos traducir.

- **Traducir:** Envía una trama de datos, donde incluye la frase o texto escrito por el usuario, a la aplicación Servidor para recoger y mostrar el resultado de la traducción. Esta trama incluye unos campos que informa a la aplicación Servidor si el usuario desea ver las palabras desconocidas por el sistema Moses (palabras que no ha podido traducirlas) y/o deletrear una palabra por si la base de datos no encuentra el signo correspondiente.

- **Ver codificación SiGML:** Muestra la secuencia de signos en SiGML obtenido en la traducción.

- **Enviar traducción al Avatar:** Crea un socket de red con el avatar y envía la secuencia de signos en SiGML.

3.3.3. Diagrama de caso de uso en la aplicación Editor

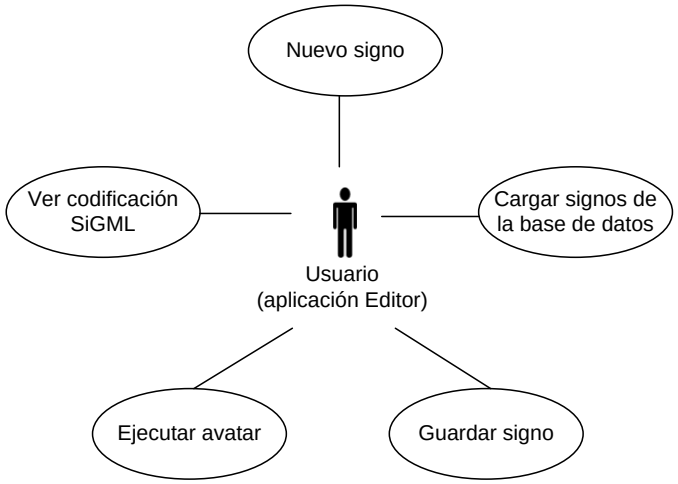


Fig. 3.9 Diagrama de caso de uso en la aplicación Editor

- **Nuevo signo:** Permite definir un nuevo signo estableciendo los movimientos de boca, cuerpo, espalda, cabeza, mirada, cejas, párpados, nariz y manos.

- **Cargar signos de la base de datos:** Establece una conexión con el servidor MySQL y muestra la lista de signos de la base de datos para cargar y editar el signo seleccionado.

- **Ver codificación del signo en SiGML:** Muestra la codificación SiGML del signo a editar.

- **Enviar signo al Avatar:** Abre un canal de comunicación con el Avatar y envía el signo seleccionado para ver sus movimientos.

- **Guardar signo en un archivo de texto:** Guarda en un archivo de texto los signos editados por el usuario. Por seguridad, hemos evitado que cualquiera pueda guardar directamente los signos a la base de datos del servidor MySQL.

3.4. Diseño de la base de datos

XAMPP, visto anteriormente en el apartado de herramientas de desarrollo, incluye el servidor MySQL, un gestor de base de datos potente y de libre distribución. Allí guardaremos los signos (glosas) en la Lengua de Signos Catalana que Barcelona Media ha ido desarrollando. Cada signo contiene la información de sus movimientos en formato SiGML. A continuación se muestra la estructura de la base de datos:

Campo	Tipo	Descripción
ID	Int(11)	Identificador del signo
GLOSA	Varchar(80)	Nombre del signo
MANOS	Text	Movimientos de manos
BOCA	Varchar(30)	Movimiento de boca
COS	Varchar(4)	Movimiento de cuerpo
ESQUENA	Varchar(4)	Movimiento de espalda
CAP	Varchar(4)	Movimiento de cabeza
MIRADA	Varchar(4)	Movimiento de mirada
CELLES	Varchar(4)	Movimiento de cejas
PARPELLES	Varchar(4)	Movimineto de párpados
NAS	Varchar(4)	Movimiento de nariz

Tabla 3.1 Diseño de la base de datos

En el campo “MANOS”, los movimientos que se ejecutan son separados por un espacio en blanco. Cada movimiento es una representación SiGML del símbolo icónico HamNoSys utilizado.

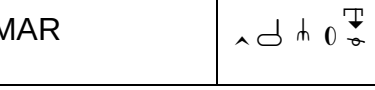
GLOSA	HamNoSys	Representación SiGML
MAR		hamfinger2 hamthumbacrossmod hamfingerhookmod hamextfingeru hampalml hamnose hammoved hamsmallmod

Tabla 3.2 Representación del signo

En el anexo D podemos ver la lista de símbolos del sistema HamNoSys con su correspondiente representación SiGML. Para la representación de los movimientos faciales (boca, cuerpo, espalda, cabeza, etc.) podemos mirar la lista de códigos del anexo E.

CAPÍTULO 4. DESARROLLO DE LAS APLICACIONES

4.1. Aplicación Servidor

El objetivo de esta aplicación es poder utilizar el sistema Moses de una manera gráfica, utilizando las herramientas que maneja para entrenar un corpus paralelo, probar su traducción y evaluar el modelo. Además, la aplicación permitirá escuchar y recibir mediante sockets de red los datos de los clientes para posteriormente enviarles a cada uno de ellos la traducción obtenida por el sistema Moses y su codificación SiGML.

- **Ejecutar un programa externo**

Entre muchas de sus funciones, la aplicación Servidor destaca por tener que acceder y ejecutar herramientas del sistema Moses. Desde Java, podemos ejecutar programas externos utilizando la clase Runtime. Ofrece diversos servicios como los de obtener la memoria disponible o ejecutar un comando del sistema con el método exec().

```
String cmd = “echo 'dilluns , 19 de novembre de 2007' | moses/moses-cmd/src/moses -f
/work/model/moses.ini > out.txt”;

String[] command = {“sh”, “-c”, cmd};
Process proc = Runtime.getRuntime().exec(command);
new Thread(){
        public void run(){
                InputStream is = proc.getInputStream();
                BufferedReader br = new BufferedReader(new InputStreamReader(is));
                String linea;
                while((linea = br.readLine()) != null){
                System.out.println(linea);
                }
        }
}.start();

new Thread(){
        public void run(){
                InputStream is = proc.getErrorStream();
                BufferedReader br = new BufferedReader(new InputStreamReader(is));
                String linea;
                while((linea = br.readLine()) != null){
                        System.out.println(linea);
                }
        }
}.start();

int returnCode = proc.waitFor();
```

En el código anterior crea un proceso de la clase Runtime para ejecutar el comando *“echo 'dilluns , 19 de novembre de 2007' | moses/moses-cmd/src/moses -f /work/model/moses.ini > out.txt”*. Acto seguido utilizamos la función *getInputStream()* para poder ver la salida estándar del comando. Para la salida de errores lo mostrará con la función getErrorStream().

La línea proc.waitFor() espera que termine el proceso. Normalmente devuelve el código 0, que significa que la ejecución ha sido correcta. En caso de que se haya producido algún error lo normal es que devuelva otro valor.

- **Conexión con una base de datos MySQL**

Otras de las funciones de la aplicación, es obtener el código SiGML de una secuencia de signos obtenida a la salida del decodificador Moses. La aplicación conectará con la base de datos para recoger la codificación SiGML del signo consultado.

Lo primero que necesitamos para conectarnos con una base de datos es un Driver. Ese Driver es la clase que, de alguna forma, sabe cómo hablar con la base de datos. Java no viene con todos los Drivers de todas las posibles bases de datos que hay.

MySQL provee conectividad para aplicaciones desarrolladas en Java mediante un driver llamado MySQL Connector/J, que es el driver JDBC (Java DataBase Connectivity) oficial para MySQL.

Lo primero que debemos hacer, es descargar la última versión del conector MySQL: http://dev.mysql.com/downloads/connector/j/. Nos bajamos y descomprimimos el archivo “mysql-connector-java-X.zip”, donde “X” es la última versión actual. El archivo que viene dentro es un archivo jar, que es donde está la clase Driver que nos interesa. Únicamente tenemos que agregarlo como una librería externa JAR a nuestra aplicación Java.

```
String ip = “localhost”;
String bd = “lsc”;
String user = “root”;
String pass = “”;

//Conexion con la base de datos
Class.forName(“com.mysql.jdbc.Driver”);
Conection con = DriverManager.getConnection(“jdbc:mysql://” + ip + “/” + bd, user, pass);

//consulta
Statement s = con.createStatement();
ResultSet rs = s.executeQuery(“SELECT * FROM signes”);

//Recorre el resultado mientras hay registros
while(rs.next()){
        System.out.println(rs.getString(0)); // muestra los valores de la 1º columna
}

//Cierra conexión
con.close();
```

En el código siguiente, la aplicación permite conectar con el servidor MySQL pasando los parámetros “dirección ip” “base de datos”, “usuario” y “contraseña”. Una vez establecida la conexión se realiza la consulta, obteniendo los resultados que necesitemos.

- **Creación de un socket TCP/IP**

La comunicación con los clientes para recibir los texto a traducir y enviarles tanto la secuencia de signos obtenida por el sistema Moses y su codificación en SiGML se realiza mediante sockets de red. Los sockets son un sistema de comunicación entre procesos de diferentes máquinas de una red. Es un punto de comunicación por el cual un proceso puede emitir o recibir información. Los procesos los tratan como descriptores de ficheros, de forma que se pueden intercambiar datos con otros procesos transmitiendo y recibiendo a través de dichos sockets.

Una de las principales características de Java es su tratamiento de la red. Java abstrae todos los detalles de manejo a bajo nivel de la red, dejándole ese trabajo a la Máquina Virtual de Java. Además, la capacidad de manejo de múltiples tareas que proporciona Java es muy cómoda a la hora de poder manejar múltiples conexiones a la vez.

El tipo de socket que utilizaremos es el Socket Stream (TCP), que son un servicio orientado a conexión, donde los datos se transfieren sin encuadrarlos en registros o bloques. Hay que establecer en primer lugar una conexión entre un par de sockets. Mientras uno de los sockets atiende peticiones de conexión (Aplicación Servidor), el otro solicita una conexión (Aplicación Cliente). Una vez que los dos sockets estén conectados, se pueden utilizar para transmitir datos en ambas direcciones.

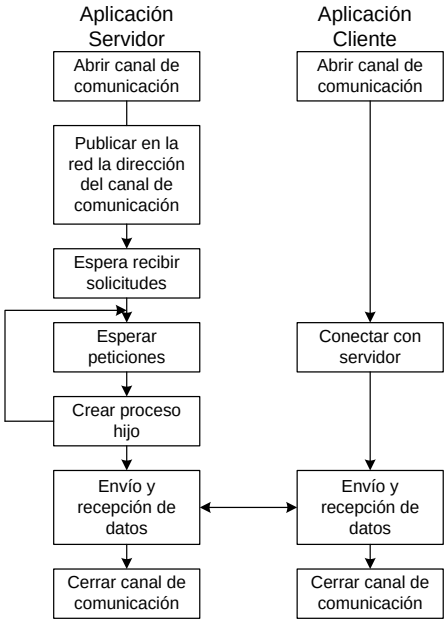


Fig. 4.1 Funcionamiento de una conexión socket TCP

Para establecer una conexión a través de un socket, tenemos que programar por un lado el servidor (aplicación Servidor) y por otro los clientes (aplicación Cliente).

Servidor

```
ServerSocket ss; // Socket servidor
Socket cliente; // Socket cliente
int port;
…
function(){
try{
        ss = new ServerSocket(port); // Crea el socket server
        //Invoca el método accept del socket servidor y devuelve una referencia al socket cliente
        while(true){
                cliente = ss.accept();
                //Obtiene una referencia a los canals de escritura y lectura del socket cliente
                DataInputStream Din = new DataInputStream(cliente.getInputStream());
                DataOutputStream Dout = new DataOutputStream(cliente.getOutputStream());
                //lee lo que escribe el socket cliente en el canal de lectura
                String datosCliente = Din.readUTF();
                String salida;
                …
                //Escribe en canal de escritura la trama de datos de salida al socket cliente
                Dout.writeUTF(salida);
                cliente.close();
        }
catch(Exception e){}
}
```

La aplicación Servidor se instala en un puerto determinado, a la espera de conexiones. Cada vez que se presenta un cliente, éste recoge la cadena de datos y envía una respuesta. En el apartado 4.2, detallaremos el código necesario en el lado cliente para conectar al socket, enviar los datos y obtener la respuesta.

Entrada y salida de datos

La aplicación escucha por un puerto determinado la entrada de datos de los clientes que se conecten. Estos datos son tramas que contiene el texto a traducir y unos campos de configuración que indica si queremos mostrar las palabras que no han podido ser traducidas por Moses y/o deletrear las palabras no encontradas en la base de datos.

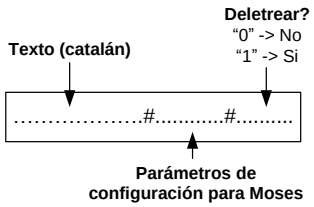


Fig. 4.2 Trama de datos del cliente

Los campos, como vemos en la figura anterior, están separados por el carácter “**#**”. De esta forma es fácil obtener la información de los campos empleando la función Split() de la clase String.

```
String entrada = “dilluns , 19 de novembre de 2007#-drop-unknown#1”;
String[] campo = entrada.split(“#”);

//campo[0] -> texto a traducir
//campo[1] -> parámetro de Moses “-drop-unknown” (no muestra las palabras desconocidas)
//campo[2] -> “1” (deletrear)
```

Una vez realizado el proceso de traducción con el sistema Mose y la obtención de la codificación SIGML de la secuencia de signos obtenida, la aplicación envía al cliente los datos de salida.

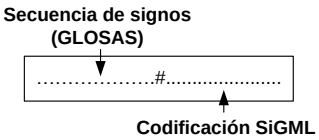


Fig. 4.3 Trama de datos del servidor

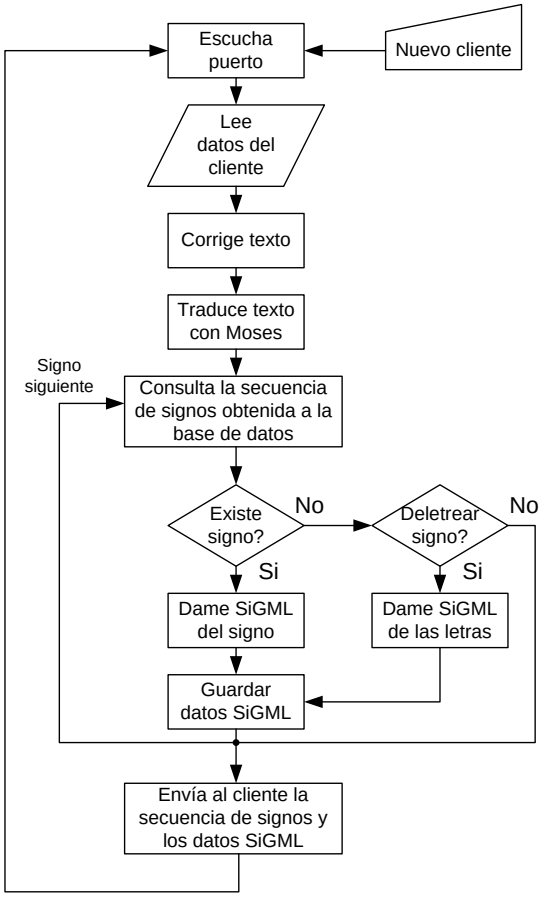


Fig. 4.4 Diagrama de flujo de entrada y salida de datos de la aplicación

Estructura de la aplicación Servidor

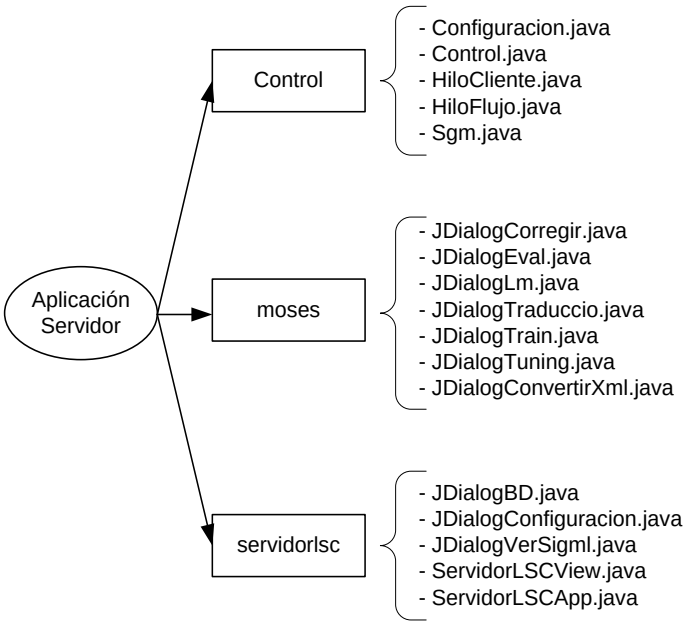


Fig 4.5 Estructura de la aplicación Servidor

- **ServidorLSCApp.java**
  Es la clase principal de la aplicación. Ejecuta el método startup() para crear un objeto de la clase ServidorLSCView.

- **ServidorLSCView.java**
  La clase ServidorView es la "ventana" principal de la aplicación.

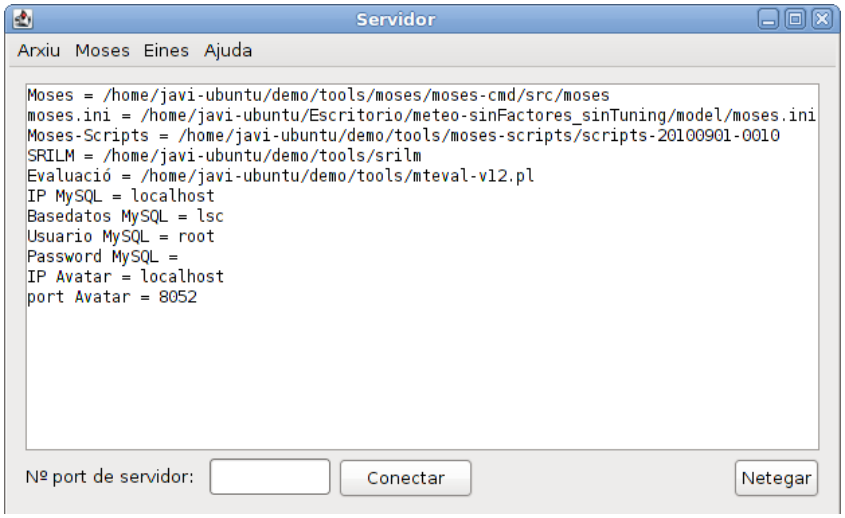


Fig. 4.6 Ventana principal de la aplicación

- **Configuracion.java**
  Es la clase donde guarda y lee las variables de las rutas de configuración de Moses, los datos del servidor MySQL y del avatar.

- **Control.java**
  La clase Control proporciona los métodos necesarios para la ejecución de las funciones que realiza la aplicación.

- **HiloCliente.java**
  La clase HiloCliente extiende de la clase Thread. Permite separar el flujo del programa cuando un cliente se conecte a la aplicación, de esta manera se crean tantos hilos como clientes se conecten.

  Esta clase llama al método run(), que es invocado cuando se inica el thread (mediante una llamada al método start()). Este método se encargará de recoger los datos del cliente, corregirlos, enviarlos al sistema Moses y devolver al cliente la traducción obtenida por el sistema Moses junto su codificación SiGML.

- **HiloFlujo.java**
  Esta clase, al igual que la clase HiloCliente permite separar el flujo del programa cuando ejecutamos un programa externo, de manera que la aplicación no quedará “colgado” mientras se esté ejecutando el proceso.

- **Sgm.java**
  La clase Sgm guarda y lee los atributos definidos para el archivo .sgm.

- **JDialogConfiguracion.java**
  Permite definir las ruta de configuración de Moses, el servidor MySQL y el avatar.

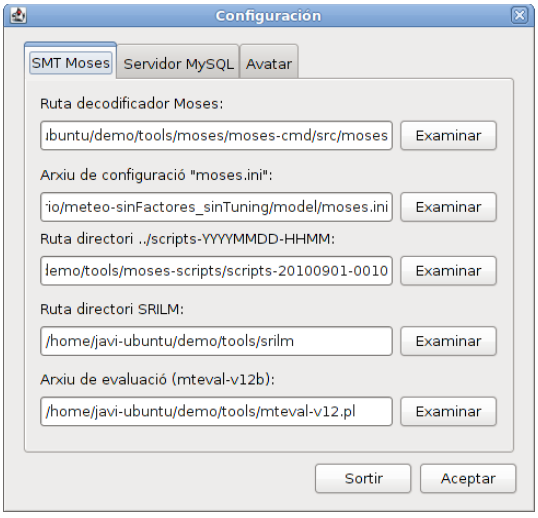


Fig. 4.7 Ventana de configuración para la aplicación Servidor

- **JDialogCorregir.java**
  Permite definir las rutas de archivos para el corpus origen y el corpus destino para ser corregidos y adaptados al formato requerido por el sistema Moses.

- **JDialogLm.java**
  Permite definir la ruta de archivo para el corpus destino y el número de n-grama. Un area de texto muestra la sintaxis final que se ejecutará.

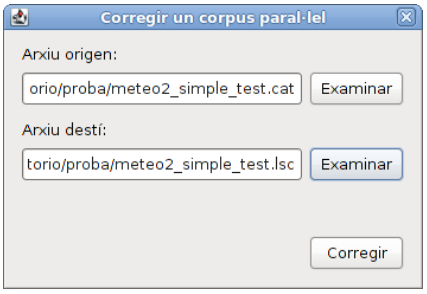


Fig. 4.8 Ventana “corregir corpus”

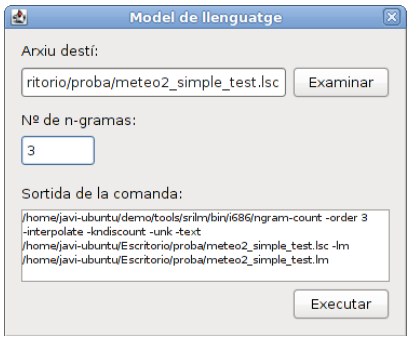


Fig. 4.9 Ventana “model de llenguatge”

- **JDialogTrain.java**
  Permite definir las rutas de archivo para el corpus origen y el corpus destino, la ruta del modelo de lenguaje, la ruta de salida del entrenamiento. Un area de texto mostrará la sintaxis final que se ejecutará. Una vez ejecutado correctamente se habrá generado el modelo de traducción y el archivo de configuración “moses.ini”.

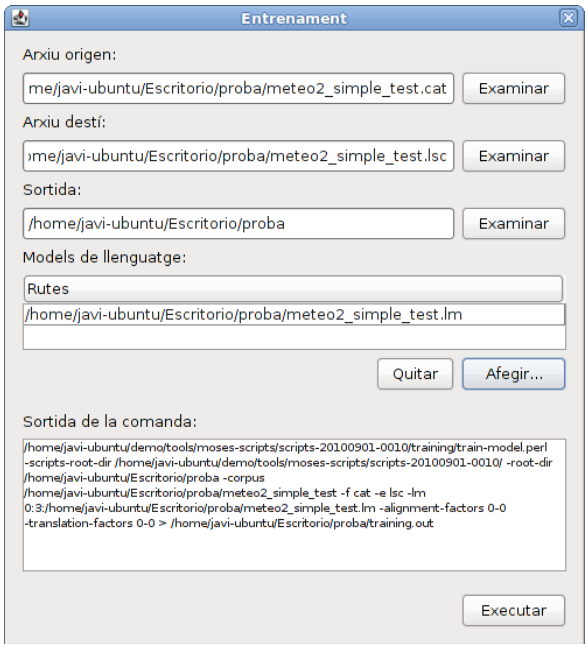


Fig. 4.10 Ventana “Entrenament”

- **JDialogTuning.java**
  Permite definir las rutas de archivos para el corpus de origen y el corpus destino, la ruta de archivo de configuración “moses.ini” generado en la etapa de entrenamiento y la ruta de salida del tuning. Genera un nuevo archivo de configuración “moses.ini” con los pesos optimizados.

- **JDialogTraduccio.java**
  Permite observar la traducción obtenida por el sistema Moses a partir de un texto de entrada. La opción “Mostrar paraules desconegudes” permite que el sistema Moses deje pasar aquellas palabras que no ha traducido. La opción “deletrear” nos indica que si un signo no se encuentra en la base de datos, la aplicación deletreará el signo obteniendo la codificación SiGML para cada letra (siempre y cuando la letra esté en la base de datos).

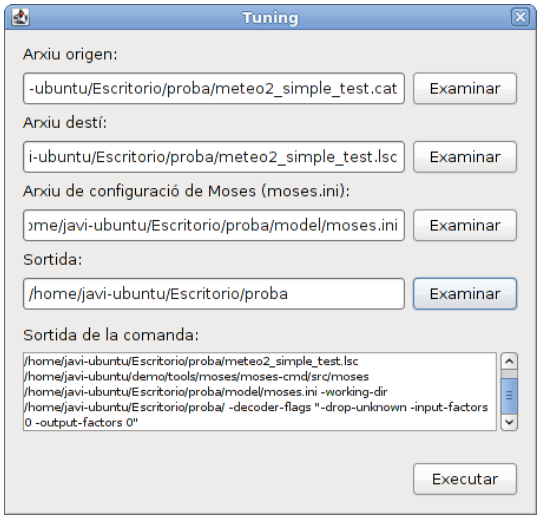


Fig. 4.11 Ventana “Tuning”

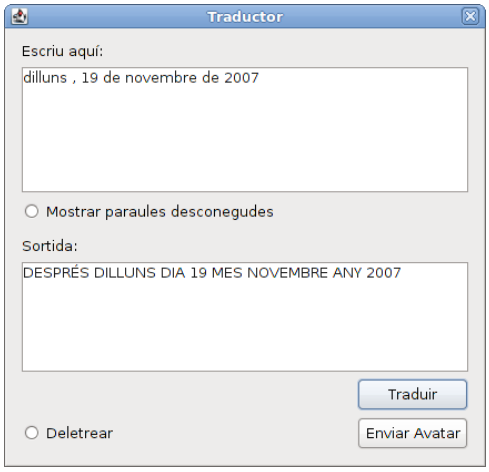


Fig. 4.12 Ventana “Traductor”

- **JDialogEval.java**
  Permite definir las rutas de archivo del formato .sgm para el corpus origen, el corpus destino (referencia) y el corpus traducido que se ha obtenido a la salida del sistema Moses a partir del corpus origen. Esto ejecutará el script de evaluación mostrando la puntuación obtenida.

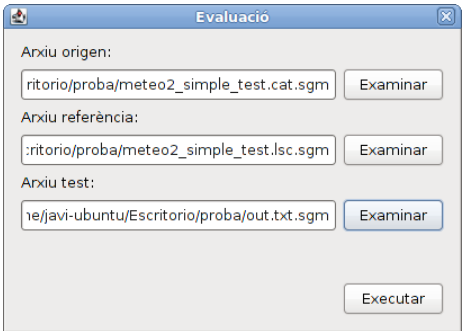


Fig. 4.13 Ventana “Evaluació”

- **JDialogConvertirXml.java**
  Permite definir la ruta de archivo de un corpus para adaptarlo al formato .sgm para la etapa de evaluación. Se especifica el tipo de corpus (origen, referencia o test) y unos atributos para el nuevo archivo .sgm.

Fig. 4.14 Ventana “Convertir .sgm”

- **JDialogBD.java**
  Permite mostrar la lista de signos de la base de datos, ver la codificación SiGML de un signo y enviarlo al avatar.

- **JDialogVerSigml.java**
  Permite mostrar la codificación SiGML del signo seleccionado de la base de datos y enviarlo al avatar.

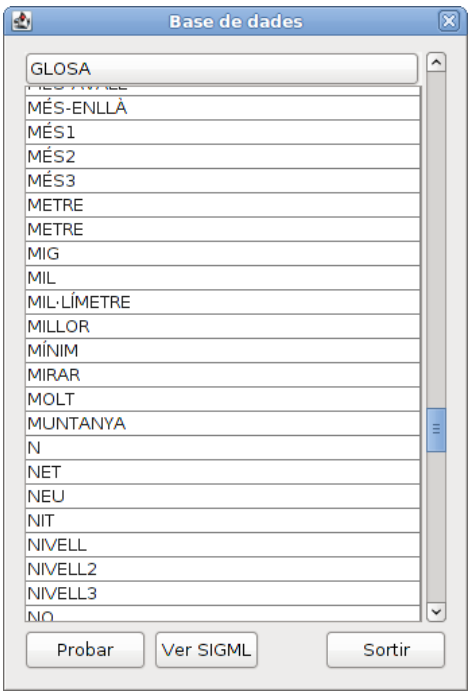


Fig. 4.15 Ventana “Base de dades” de la aplicación Servidor

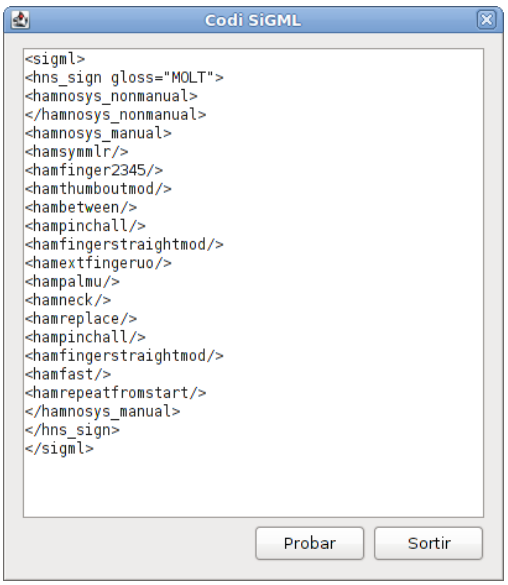


Fig. 4.16 Ventana “Codi SiGML”

4.2. Aplicación Cliente

El objetivo de esta aplicación es permitir que un usuario pueda traducir un texto (en catalán) a la lengua de signos catalana obteniendo su correspondiente secuencia de signos en glosas y su codificación SiGML.

La aplicación Cliente conectará mediante un socket de red a la aplicación Servidor para enviarle el texto a traducir y esperará obtener una respuesta por parte del Servidor. Una vez obtenida la respuesta se mostrará por pantalla la traducción en glosas, guardando en una variable su codificación SiGML. La aplicación Cliente crea un nuevo socket de red con el avatar para enviar los datos SiGML obtenidos.

- **Conectar con un socket de red**

Una de sus funciones más importantes es la conexión con la aplicación Servidor y con el avatar a través de un socket de red. A continuación vamos a mostrar como conectar con un socket de red.

```
Socket cliente;
String ip;
int port;
…
function(){
try{
        cliente = new Socket(ip, port); //Crea una conexión al socket servidor
        //Crea las referencias al canale de escritura y lectura del socket
        OutputStream out = cliente.getOutputStream();
        DataOutputStream Dout = new DataOutputStream(out);
        InputStream in = cliente.getInputStream();
        DataInputStream Din = new DataInputStream(in);
        String entrada;
        …
        Dout.writeUTF(entrada); //Escribe en el canal de escritura del socket
        String salida = Din.readUTF(); //Espera la respuesta por el canal de lectura
        cliente.close();
}catch(Exception e){}
}
```

La aplicación cliente se conecta con la aplicación Servidor indicando el nombre de la máquina y el número puerto en el que el servidor está instalado. Una vez conectado, el cliente envía una cadena de datos al servidor y recibe una respuesta.

Lo mismo pasa al enviar datos a nuestro Avatar. El avatar se muestra como un servidor que escucha por su número de puerto 8052, pero únicamente recibe datos, sin devolver nada.

Estructura de la aplicación Cliente

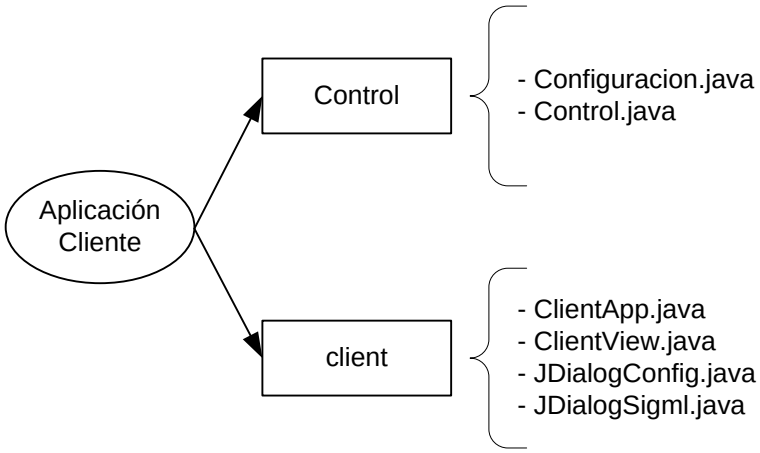


Fig. 4.17 Estructura de la aplicación Cliente

- **ClientApp.java**
  Es la clase principal de la aplicación. Ejecuta el método startup() para crear un objeto de la clase ClientView.

- **ClientView.java**
  La clase ClientView es la "ventana" principal de la aplicación.

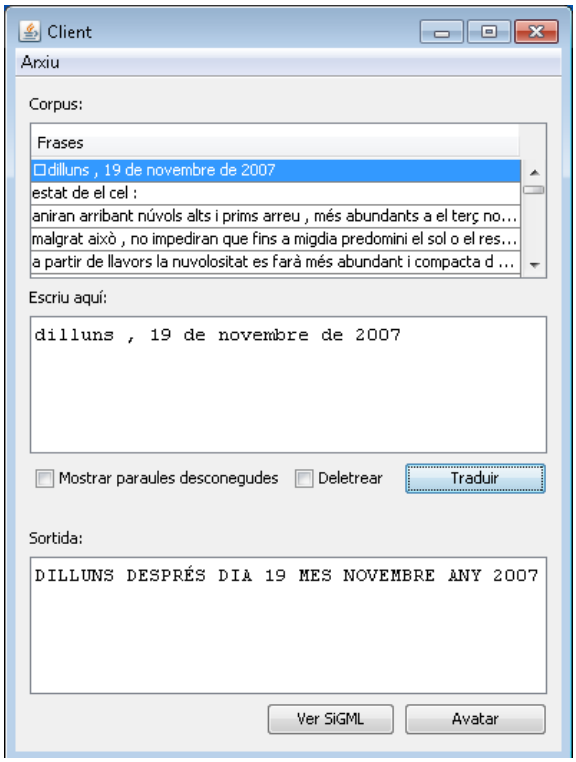


Fig. 4.18 Ventana principal de la aplicación Cliente

- **Configuracion.java**
  La clase Configuracion permite guardar y leer las rutas de las herramientas del sistema Moses, los datos de configuración del servidor MySQL y del avatar.

- **Control.java**
  La clase Control proporciona los métodos necesarios para la ejecución de las funciones que realiza la aplicación.

- **JDialogConfig.java**
  Permite definir las rutas de las herramientas del sistema Moses, los datos de configuración del servidor MySQL y del avatar.

- **JDialogSigml.java**
  Permite visualizar la secuencia de signos en SiGML obtenidos en la traducción.

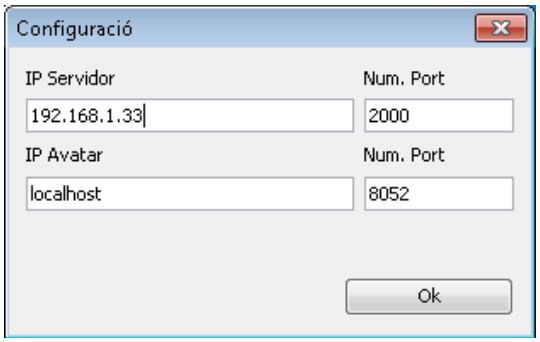


Fig. 4.19 Ventana de configuración para la aplicación Cliente

4.3. Aplicación Editor

El objetivo de esta aplicación es permitir crear o editar un signo de la base de datos. Además de definir el movimiento de manos, podremos añadir movimientos faciales (boca, cuerpo, espalda, cabeza, mirada, cejas, párpados y nariz).Los movimientos creados por la aplicación se pueden exportar a SiGML, de manera que podemos enviar el signo editado a nuestro avatar.

Por último, los signos editados pueden ser guardados en un archivo de texto, no directamente de la base de datos.

Estructura de la aplicación Editor

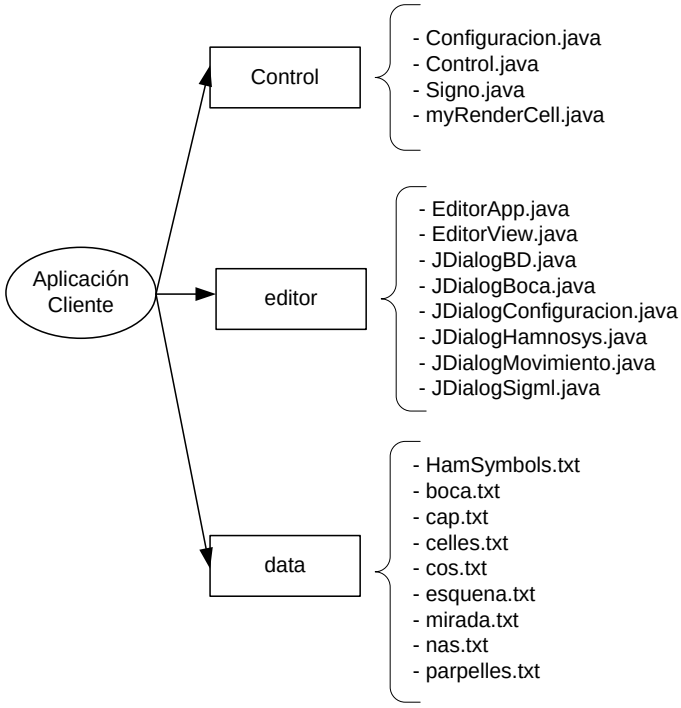


Fig. 4.20 Estructura de la aplicación Editor

- **EditorApp.java**
  Es la clase principal de la aplicación. Ejecuta el método startup() para crear un objeto de la clase EditorView.

- **EditorView.java**
  La clase EditorView es la “ventana” principal de la aplicación.

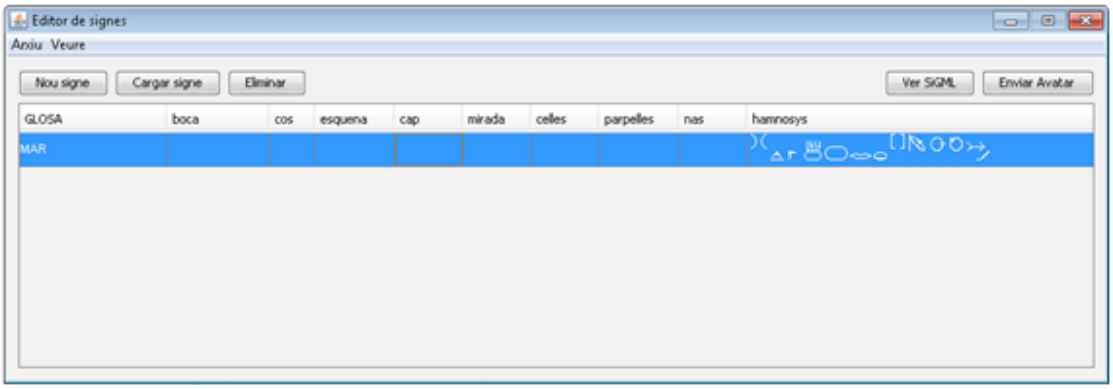


Fig. 4.21 Ventana principal de la aplicación Editor

- **Configuración.java**
  La clase Configuracion permite guardar y leer los datos de configuración del servidor MySQL y del avatar.

- **Control.java**
  La clase Control proporciona los métodos necesarios para la ejecución de las funciones que realiza la aplicación.

- **Signo.java**
  La clase Signo permite guardar y leer los valores para los movimientos de boca, cuerpo, espalda, cabeza, mirada, cejas, párpados, nariz, manos y el nombre del signo (glosa).

- **myRenderCell.java**
  Es una clase que extiende de la clase DefaultTableCellRenderer. Contiene el método getTableCellRendererComponent que permite definir las propiedades de las columnas de la tabla que queramos editar. Esta clase la utilizaremos para avisar que la última columna (“HamNoSys”), tendrá el tipo de letra “HamNoSysUnicode.ttf”, de esta manera podremos ver los icónos gráficos sin ningún problema.

- **JDialogConfiguracion.java**
  Permite definir los datos de configuración del servidor MySQL y del avatar.

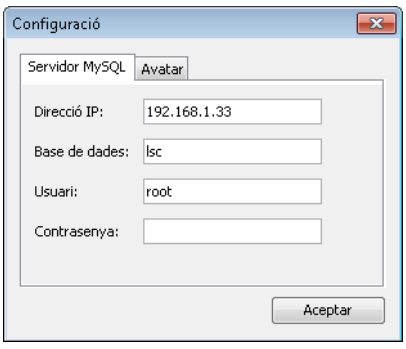


Fig. 4.22 Ventana de configuración de la aplicación Editor

- **JDialogBD.java**
  Muestra la lista de signos de la base de datos.

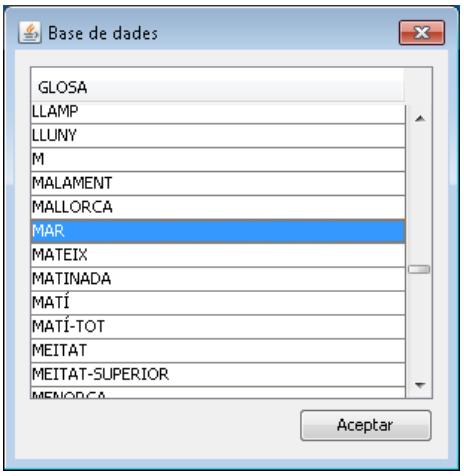


Fig. 4.23 Ventana “base de dades” de la aplicación Editor

- **JDialogBoca.java**
  Lee la lista de códigos del archivo “boca.txt” para definir el tipo de movimiento de boca. Los movimientos pueden ser: dientes, mandíbula, labios, lengua y mejilla,

- **JDialogMovimiento.java**
  Dependiendo del tipo de movimiento que queramos editar, la aplicación leerá el archivo de texto correspondiente para cargar la lista de movimientos posibles para cada tipo (boca.txt, cap.txt, celles.txt, cos.txt, esquena.txt, mirada.txt, nas.txt o parpelles.txt).

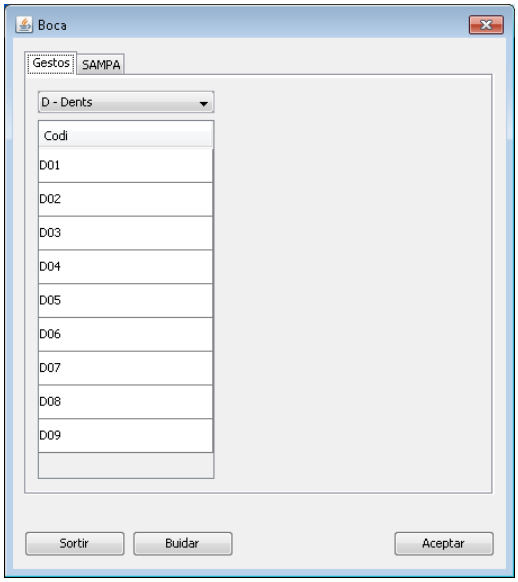


Fig. 4.24 Lista de movimientos para la boca

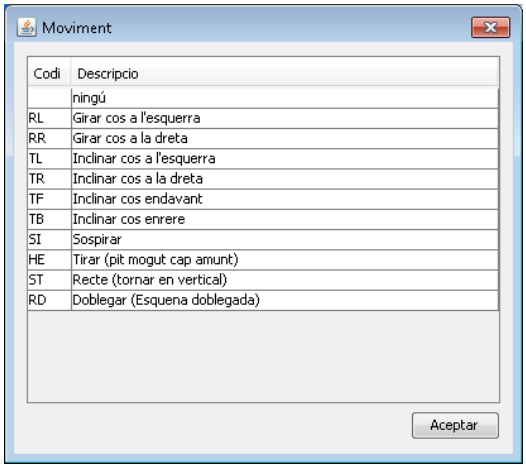


Fig. 4.25 Lista de movimientos para el cuerpo

- **JDialogHamnosys.java**
  Permite definir el movimiento de manos. Cada botón representa un icóno gráfico del sistema de notación HamNoSys. Los iconos están divididos según la forma de mano, la orientación, la localización, el movimiento y si el movimiento es con la mano secundaria.

- **JDialogSigml.java**
  Muestra la codificación SiGML del signo que estemos editando.

- **HamSymbol.txt**
  Lista de iconos HamNoSys con su correspondiente traducción textual para la codificación SiGML (ver anexo D).

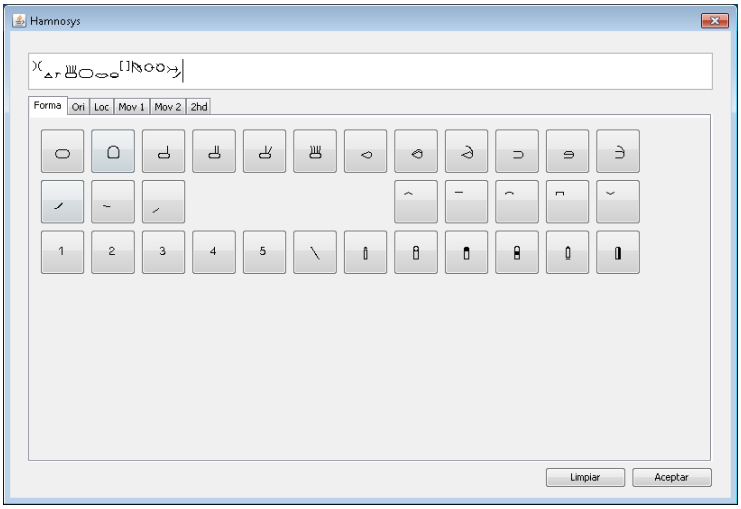


Fig. 4.26 Ventana “HamNoSys” para la configuración de manos

CAPÍTULO 5. CONCLUSIONES

5.1. Pruebas y resultados

La información utilizada para los experimentos consiste en un corpus paralelo que contiene 262 frases típicas de un texto restringido en el dominio de la meteorología. Este conjunto de fases se dividió en 2 grupos de forma arbitraria: entrenamiento (contenido aproximadamente el 80% de las frases) y evaluación (con el 20% de las frases). A continuación se muestra un resumen de los datos:

		Catalán	**LSC**
Total	Pares de frases	262	
	Nº de palabras / glosas	3536	2440
Entrenamiento	Pares de frases	221	221
	Nº de palabras / glosas	3067	2069
Test	Pares de frases	41	41
	Nº de palabras / glosas	469	371

Tabla 5.1 Datos de entrenamiento y test

Para este experimento se analizó la influencia del tipo de alineamiento utilizado en la etapa de entrenamiento para la evaluación del corpus paralelo.

Se emplean métricas (NIST y BLEU) que compara la traducción que realiza el sistema con una traducción de referencia. Los resultados para cada tipo de alineamiento pueden verse en la tabla 5.2 y en la figura 5.1.

Tipo de alineamiento	**Test**	
	NIST	**BLEU**
Origen-Destino (OD)	3.5186	0.1698
Destino-Origen (DO)	4.2156	0.2220
Intersección (I)	3.7009	0.1835
Unión (U)	2.4214	0.1277
Crecimiento (C)	3.3626	0.1798
Crecimiento diagonal (CD)	3.6635	0.1715
Crecimiento diagonal evitando casos de no alineamiento (CDP)	3.5186	0.1698

Tabla 5.2 Resultados con cada tipo de alineamiento en el modelo de traducción

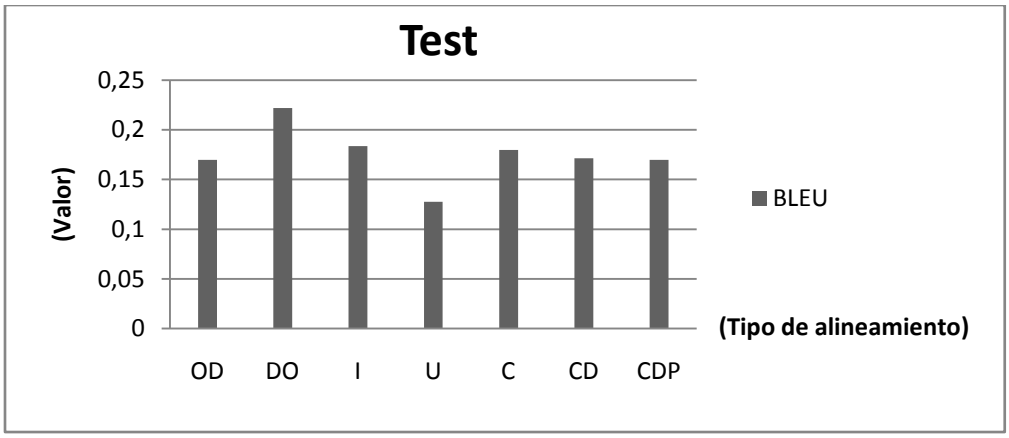


Fig. 5.1 Comparativa de los valores obtenido en traducción con cada tipo de alineamiento por BLEU

Observamos que el tipo de alineamiento que mejor resultado se ha obtenido es el tipo Destino-Origen. Esto es así porque, al traducir a LSC una frase en catalán, lo que se está haciendo principalmente es extraer información semántica, y esto se consigue mejor con el alineamiento destino-origen (DO).

Es importante tener en cuenta que este comportamiento se da con un corpus limitado. Al aumentar el corpus, los resultados podrían variar.

Por último, en el archivo de configuración moses.ini, los distintos modelos implicados en el proceso de traducción (modelo de traducción y de lenguaje de la lengua destino) tienen unos pesos por defecto que no son los óptimos. Utilizando previamente la etapa "Tuning" mejoramos ligeramente los resultados obtenidos de la tabla 5.2.

Tipo de alineamiento	**Test**			
	Sin Tuning		**Con Tuning**	
	NIST	BLEU	NIST	BLEU
Destino-Origen	4.2156	0.2220	4.3991	0.2446

Tabla 5.3 Resultados obtenidos sin/con Tuning para el alineamiento Destino-Origen

La mejora conseguida ha sido de una reducción relativa de la tasa de error del 10%.

5.2. Limitaciones

Una de las principales limitaciones que hemos encontrado es la escasez de textos en la lengua de signos catalana, disponiendo de un corpus muy limitado. Esta es una de las razones obvias por el que el resultado de la traducción no sea del todo correcto. Tener un gran volumen de textos requiere la contratación de intérpretes y eso supone un coste adicional de capital humano.

Otra de las limitaciones y que solucionarlo conllevaría mucho tiempo es la creación de una extensa y completa base de datos con todos los signos existentes de la lengua catalana.

En cuanto al sistema Moses, cabe remarcar que su ejecución solo es posible si se está corriendo en un entorno Linux. Esto nos delimita que Moses no es independiente del sistema operativo instalado y obliga a que nuestra la aplicación Servidor deba estar instalada en el mismo entorno.

Por último, y no tan drástico, no hemos de olvidar que el avatar escogido para este proyecto únicamente funciona en los sistemas operativos (XP, Vista, 7) y en las últimas versiones del MAC OS X 10.5 y 10.6. Actualmente no es compatible con Linux.

5.3. Líneas futuras

Este apartado abre camino algunas ideas que podrían desarrollar posibles mejoras a implementar sobre el proyecto ya realizado.

- Integrar diferentes corpus paralelos dependiendo del dominio utilizado (política, deporte, sanidad, economía, etc.).

- Añadir un módulo de reconocimiento de voz que permita pasar a un texto las frases pronunciadas por el usuario.

- Mejorar los resultados de la traducción añadiendo información lingüística en el corpus paralelo. Moses permite trabajar con modelos que incorporan factores lingüísticos.

- Agregar en las glosas parámetros que permitan incluir rasgos no manuales, es decir, movimientos del cuerpo y la cara.

- Añadir nuevos signos de la lengua catalana en la base de datos. Actualmente disponemos unos 260 signos aproximadamente.

- Disposición de un Avatar que pueda ejecutarse independientemente del sistema operativo que se encuentre.

5.4. Conclusiones personales

La primera parte de la elaboración de este proyecto, y la que más dificultades tuve, fue la instalación, configuración y funcionamiento del sistema estadístico Moses. Se necesitó entender los pasos a seguir para la elaboración de un modelo de traducción utilizando un corpus paralelo, de cómo ejecutar y obtener la traducción desde una aplicación Java. Durante ese tiempo adquirí nuevos conocimientos del lenguaje java.

Por otro lado aprendí, con ayuda de amigos sordos signantes, a transcribir en glosas textos a la lengua de signos, conociendo su estudio lingüístico para formar correctamente las frases. Este paso era importante para la elaboración del corpus paralelo.

Una segunda parte de la dedicación de este proyecto fue comprender el funcionamiento del avatar, de cómo y de qué manera pudiésemos enviar los signos para que procesara y ejecutara sus movimientos. Para ello se estudió el formato SiGML, una codificación de los signos mediante el sistema de notación HamNoSys el cual el avatar era capaz de leerlo y ejecutarlo. Se diseñó una base de datos donde se pudo almacenar unos 260 signos con su correspondiente codificación en SiGML.

El resultado de esta pequeña plataforma ha sido muy satisfactorio, a pesar de tener un corpus limitado y de no poder traducir cualquier texto. Tradicionalmente se ha relacionado la calidad de la traducción con el conocimiento lingüístico. Cuando más conocimiento lingüístico tenga un sistema, mejores serán las traducciones; pero es evidente que eso significa más dinero para comprarlo, ya que la construcción de un sistema con conocimiento lingüístico exige una gran inversión tecnológica y en capital humano. Con la aparición de los sistemas estadísticos, que no suponen unos costes tan elevados y que traducen bastante bien -sobre todos cuando las lenguas son cercanas- parece que la relación calidad-precio no siempre ha de ser tan directa.

La traducción automática no sustituirá a la traducción humana. Tiene unas limitaciones que difícilmente podrán ser superadas. Ahora bien, se debe reconocer que la traducción humana tiene unas limitaciones que la traducción automática no tiene. Un traductor humano no traduce tan rápidamente cantidades ingentes de documentos ni siempre está disponible. Principalmente, es de ayuda cuando hay que traducir muchos documentos en un corto periodo de tiempo.

La elaboración de este proyecto permitiría a los sordos signantes poder acceder con más facilidad a la información. Podíamos instalar este tipo de sistema en servicios de transporte, como por ejemplo en el metro, donde la información por megafonía es una barrera de comunicación para los sordos signantes.

En cuanto el estudio de ambientación, podemos afirmar que el proyecto no ha tenido un impacto relevante. Esto es debido a que todo lo que se ha hecho ha estado utilizando el ordenador y ningún otro material o escenario adicional.

BIBLIOGRAFÍA

[1] Moses Translation System
http://www.statmt.org/moses/

[2] Virtual Humans Research at UEA
http://vh.cmp.uea.ac.uk/index.php/Main_Page

[3] ViSiCAST project at UEA.
http://www.visicast.cmp.uea.ac.uk/Visicast_index.html

[4] eSIGN project at UEA
http://www.visicast.cmp.uea.ac.uk/eSIGN/index.html

[5] STAR Laboratory: SRI Language Modeling Toolkit
http://www-speech.sri.com/projects/srilm/

[6] GIZA++ statistical translation models toolkit
http://code.google.com/p/giza-pp/

[7] Avatar JASigning
http://vhg.cmp.uea.ac.uk/tech/jas/095f/

[8] Bibliografía de William C. Stokoe
http://gupress.gallaudet.edu/stokoe.html

[9] Schmaling, C. et al. HamNoSys 4.0, University of Hamburg.
http://www.sign-lang.uni-hamburg.de/projekte/hamnosys/hns4.0/hns4.0eng/contents.html

[10] SIGNApuntes - Apuntes sobre la Lengua de Signos y el ejercicio profesional de su interpretación.
http://signapuntes.8forum.info/forum

[11] La lengua natural de las personas sordas
http://www.lenguadesignos.org/

[12] La cultura sorda
http://cultura-sorda.eu/index.html

[13] Signem. Guía básica para la comunicación en lengua de signos catalana
http://philipjfry.cephis.uab.cat/signem/index.php?idioma=es&plantilla=portada

[14] FESOCA (Federació de Persones Sordes de Catalunya).
http://www.fesoca.org/

[15] CNSE (Confederación Estatal de Personas Sordas).
http://www.cnse.es/lengua_signos/lengua_sing.html

[16] Alfabeto fonético SAMPA
http://www.phon.ucl.ac.uk/home/sampa/

[17] Java Web Start Guide
http://download.oracle.com/javase/1.5.0/docs/guide/javaws/developersguide/contents.html

[18] Apache Software Foundation
http://www.apache.org/

[19] MySQL
http://www.mysql.com/

[20] apache friends XAMPP
http://www.apachefriends.org/es/xampp.html

[21] Lenguaje SQL en W3schools
http://www.w3schools.com/sql/default.asp

[22] Froufe, A., "Java 2 Manual de usuario y tutorial", Edición RA-MA, Madrid, 2005.

[23] Oliver, A. Moré, J. "Traducción y tecnologías", Barcelona UOC, Universitat Oberta de Catalunya, 2008.

[24] Edinburgh System Description for the 2005 IWSLT Speech Translation Evaluation, Philipp Koehn, Amittai Axelrod, Alexandra Birch, Chris Callison-Burch, Miles Osborne and David Talbot, International Workshop on Spoken Language Translation 2005.

[25] Moses: Open Source Toolkit for Statistical Machine Translation, Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin*, Evan Herbst*, ACL 2007, demonstration session

[26] Valenzuela, J., "Programando linguistic: (un prototipo de traducción automática inglés-español)", Murcia, Universidad de Murcia, 2000.

[27] Martí, M., Llisterri, J., "Tecnologías del texto y del habla", Barcelona, Ediciones Univeristat de Barcelona, 2004.

ANEXOS

TÍTOL DEL TFC: Creación de una interfaz gráfica de traducción automática para las lenguas de signos.

TITULACIÓ: Ingeniería Técnica de Telecomunicación, especialidad Sistemas de Telecomunicación.

AUTOR: Javier Marín Rey

DIRECTOR: Dolors Royo

SUPERVISOR: Guillem Massó (Barcelona Media)

DATA:

ANEXO A. GUÍA DE INSTALACIÓN DEL SMT MOSES

Pre-requisitos

La instalación del SMT Moses[1] se ejecutará desde Linux. Antes de empezar con la instalación es importante asegurar de que tengamos instalados los siguientes paquetes que se pueden descargar desde el Administrador de paquetes Synaptic:
Instalar desde el Administrador de paquetes Synaptic los siguientes paquetes:

- Gcc
- G++
- Make
- Gawk
- Gzip
- Tcl8.5
- Tcl8.5-dev
- Csh
- Autoconf
- Automake 1.9
- Texinfo
- Zlib1g
- Zlib1g-dev
- Zlib-bin
- Zlibc
- Libtool

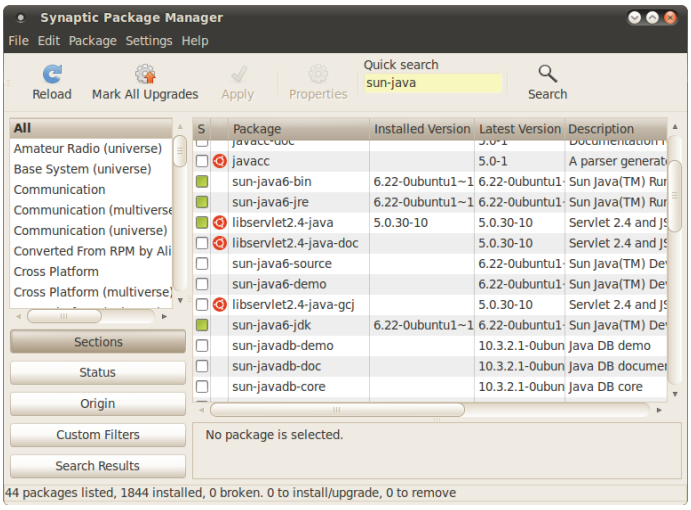


Fig. 1 Administrador de paquetes Synaptic de Linux

[1] http://www.statmt.org/moses/

Instalación Giza++

Descargar y descomprimir en la carpeta tools/ la herramienta GIZA++[2]
http://code.google.com/p/giza-pp/downloads/list

Antes de compilar debemos modificar el archivo file_spec.h, situado dentro de la carpeta giza-pp/GIZA++/

Aquí dejo un ejemplo de lo que debe contener ese archivo:

```
#ifndef FILE_SPEC_H
#define FILE_SPEC_H
#include <time.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

char *Get_File_Spec (){

  struct tm *local;
  time_t t;
  char *user;
  char time_stmp[19];
  char *file_spec = 0;

  t = time(NULL);
  local = localtime(&t);

  sprintf(time_stmp, "%04d-%02d-%02d.%02d%02d%02d.", 1900 + local->tm_year, (local->tm_mon + 1), local->tm_mday, local->tm_hour, local->tm_min, local->tm_sec);

  user = getenv("USER");

  file_spec = (char *)malloc(sizeof(char) *
                      (strlen(time_stmp) + strlen(user) + 1));

  file_spec[0] = '\0';
  strcat(file_spec, time_stmp) ;
  strcat(file_spec, user);
  return file_spec;
}
#endif
```

Una vez modificado el archivo ya podremos compilar GIZA++

```
$ make
```

Se generará unos archivos ejecutables (GIZA++, mkcls, snt2cooc.out) donde los copiaremos en una nueva carpeta llamada por ejemplo “/bin”.

[2] http://code.google.com/p/giza-pp/

```
$ mkdir tools/bin
$ cd tools
$ cp giza-pp/GIZA++-v2/GIZA++ bin/
$ cp giza-pp/mkcls-v2/mkcls bin/
$ cp giza-pp/GIZA++-v2/snt2cooc.out bin/
```

Instalación SRILM

Descargar y descomprimir en la carpeta tools/ la herramienta SRILM[3]
http://www-speech.sri.com/projects/srilm/download.html

Antes de compilarlo, editaremos el fichero Makefile. Es posible que este fichero no tenga permisos de escritura, para ello escribimos:

```
$ chmod +w Makefile
```

Dentro del archivo Makefile editamos la ruta del SRILM:

```
SRILM = /home/javier.marin/demo/tools/srilm
```

Abrimos el archivo Makefile.machine.XXX que está dentro de la carpeta common/, donde "XXX" representa el valor del tipo de máquina. Podemos comprobar que tipo de máquina que tenemos con la siguiente comanda:

```
$ ./srilm/sbin/machine-type
```

Sobre ese archivo reemplazamos las siguientes líneas:

```
# Tcl support (standard in Linux)
TCL_INCLUDE = -I/usr/include/tcl8.5/
TCL_LIBRARY = -L/usr/lib/libtcl8.5.so
```

A continuación compilamos:

```
$ sudo make
$ sudo make World
```

[3] http://www-speech.sri.com/projects/srilm/

Instalación Moses

Descargar y descomprimir en la carpeta tools/ el decodificador Moses
http://sourceforge.net/projects/mosesdecoder/files/

Una vez descomprimido, ejecutamos los siguientes scripts:

```
$ ./regenerate-makefiles.sh
$ ./configure --with-srilm=/home/javi-ubuntu/demo/tools/srilm
```

Y compilamos:

```
$ make -j 4
```

Scripts de apoyo de Moses

Moses usa una serie de scripts de soporte para el entrenamiento, tunning y otras tareas. Creamos una carpeta moses-scripts/. Editamos las rutas siguientes del archivo Makefile de la carpeta ../moses/scripts/

```
TARGETDIR?=/home/usuario/demo/tools/moses-scripts
BINDIR?=/home/usuario/demo/tools/bin
```

La primera línea define donde está la carpeta que habíamos creado anteriormente y en la segunda línea define la ruta de los archivos ejecutables del GIZA++.

```
$ cd moses/scripts
$ make release
```

Una vez compilado se habrá creado una carpeta del tipo “scripts-YYYYMMDD-HHMM” dentro de la carpeta moses-scripts/.

Por último abriremos el archivo train-model.perl situado en la carpeta scripts-YYYYMMDD-HHMM/training/. Este archivo contiene una serie de funciones para ejecutar el entrenamiento de un corpus paralelo. La función que debemos modificar se llama reduce_factors(), a continuación dejo un ejemplo de lo que debe contener esa función:

```
sub reduce_factors {
    my ($full,$reduced,$factors) = @_;
    print STDERR "(1.0.5) reducing factors to produce $reduced  @ ".`date`;
    while(-e $reduced.".lock") {
      sleep(10);
    }
    if (-e $reduced) {
        print STDERR "  $reduced in place, reusing\n";
        return;
    }
```

```
    `touch $reduced.lock`;
    # my %INCLUDE;
    # foreach my $factor (split(/,/,$factors)) {
      # $INCLUDE{$factor} = 1;
    # }
    my @INCLUDE = sort {$a <=> $b} split(/,/,$factors);

    *IN = open_or_zcat($full);
    open(OUT,">".$reduced) or die "ERROR: Can't write $reduced";
    my $nr = 0;
    while(<IN>) {
        $nr++;
        print STDERR "." if $nr % 10000 == 0;
        print STDERR "($nr)" if $nr % 100000 == 0;
      chomp; s/ +/ /g; s/^ //; s/ $//;
      my $first = 1;
      foreach (split) {
          my @FACTOR = split /\Q$___FACTOR_DELIMITER/;
              # \Q causes to disable metacharacters in regex
          print OUT " " unless $first;
          $first = 0;
          my $first_factor = 1;
            foreach my $outfactor (@INCLUDE) {
              print OUT "|" unless $first_factor;
              $first_factor = 0;
              my $out = $FACTOR[$outfactor];
              die "ERROR: Couldn't find factor $outfactor in token \"$_\" in
$full LINE $nr" if !defined $out;
              print OUT $out;
            }
          # for(my $factor=0;$factor<=$#FACTOR;$factor++) {
            # next unless defined($INCLUDE{$factor});
            # print OUT "|" unless $first_factor;
            # $first_factor = 0;
            # print OUT $FACTOR[$factor];
          # }
      }
      print OUT "\n";
    }
    print STDERR "\n";
    close(OUT);
    close(IN);
    `rm -f $reduced.lock`;
}
```

ANEXO B. FORMATO DE ARCHIVO .SGM

Formato de archivo de origen

Contiene la etiqueta “<srcset>” y los siguientes atributos:

- “setid” : El conjunto de datos
- “srclang”: La lengua de origen

La etiqueta “<srcset>” contiene uno o más elementos “doc”, que tendrá las siguientes atribuciones:

- “docid”: El documento
- “genre”: El tipo de dato

Cada elemento “doc” contiene varios segmentos (elementos “seg”). Cada segmento tiene un atributo único, “id”, que debe incluir el uso de comillas dobles.

Ejemplo:

```
<srcset setid="meteo" srclang="any">
<DOC docid="meteo">
<seg id="1"> dijous , 22 de novembre de 2007</seg>
…
</DOC>
</srcset>
```

Formato de archivo de referencia

Contiene una o más etiquetas “<refset>”. Cada etiqueta “<refset>” contiene los siguientes atributos:

- “setsid”: El conjunto de datos
- “srclang”: La lengua origen
- “trglang”: La lengua destino
- “refid”: La referencia actual

El formato de los elementos del documento es exactamente igual como el archivo de origen descrito anteriormente.

Ejemplo:

```
<refset setid="meteo" srclang="any" trglang="lsc">
<DOC docid="meteo" sysid="ref">
<seg id="1"> dijous després dia 22 mes novembre any 2007</seg>
…
</DOC>
</refset>
```

Formato de archivo de test

Un archivo de traducción contiene una o más etiquetas “<tstset>”. Cada “<tstset>” contiene los siguientes atributos:

- “setid”: El conjunto de datos
- “srclang”: La lengua origen
- “trglang”: La lengua destino
- “sysid”: Un nombre de identificación del sitio y sistema

El contenido de cada “<tstset>” es exactamente igual que los archivos de origen y referencia.

Ejemplo:

```
<tstset setid="meteo" srclang="any" trglang="lsc" sysid=”meteo”>
<DOC docid="meteo" sysid="ref">
<seg id="1"> dijous després dia 22 mes novembre any 2007</seg>
…
</DOC>
</tstset>
```

ANEXO C. FORMATO DE ARCHIVO .JNLP

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+" codebase="http://localhost/web-lsc/" href="editor.jnlp">
      <information>
            <title>Editor</title>
            <vendor>xmarin</vendor>
            <homepage href="http://localhost/" />
            <description>Editor</description>
      </information>
      <resources>
            <j2se version="1.6+" />
            <jar href="editor.jar" />
      </resources>
      <offline-allowed />
      <security>
            <all-permissions/>
      </security>
      <application-desc
         main-class="editor.EditorApp">
     </application-desc>
</jnlp>
```

Un archivo JNLP es un XML especialmente formado compuesto por:

- Una cabecera XML típica.
- Una ruta predeterminada para que los archivos puedan ser llamados desde un path relativo.
- Una o varias etiquetas “information” en que van varias informaciones.
- Una etiqueta “resources”.
- Una etiqueta “security”.
- Una etiqueta “aplication-desc” con la clase predeterminada a ejecutar.

El ejemplo mostrado no incluye todas las posibles opciones. Hay muchas más que pueden verse desde la documentación de Java:
http://download.oracle.com/javase/6/docs/technotes/guides/javaws/

ANEXO D. LISTA DE SÍMBOLOS HAMNOSYS

Representación de los símbolos HamNoSys en forma icónica y textual para la codificación SiGML.

	hamalternatingmotion		hamextfingerr
	hamarmextended		hamextfingeru
	hambelowstomach		hamextfingerui
	hamcee12		hamextfingerul
	hamceeall		hamextfingeruo
	hamceeopen		hamextfingerur
	hamcheek		hameyebrows
	hamchest		hameyes
	hamchin		hamfinger2
	hamclose		hamfinger23
	hamear		hamfinger2345
	hamearlobe		hamfinger23spread
	hamelbowinside		hamfingerbase
	hamextfingerd		hamfingermidjoint
	hamextfingerdi		hamfingernail
	hamextfingerdl		hamfingerpad
	hamextfingerdo		hamfingertip
	hamextfingerdr		hamfist
	hamextfingeri		hamflathand
	hamextfingeril		hamforehead
	hamextfingerir		hamfusionbegin
	hamextfingerl		hamfusionend
	hamextfingero		hamhandback
	hamextfingerol		hamhead
	hamextfingeror		hamheadtop

2	hamindexfinger
⌔	hamlips
⌐	hamlowerarm
3	hammiddlefinger
⌡⌠	hamneck
⍁	hamnondominant
⍋	hamnose
⍋	hamnostrils
⌣	hampalm
⌔	hampalmd
⌔	hampalmdl
⌔	hampalmdr
⌔	hampalml
⌔	hampalmr
⌔	hampalmu
⌔	hampalmul
⌔	hampalmur
[	hamparbegin
]	hamparend
⌔	hampinch12
⌔	hampinch12open
⌔	hampinchall
5	hampinky
5|	hampinkyside
⌿	hamplus
4	hamringfinger
(	hamseqbegin
)	hamseqend
▽	hamshoulders
▽	hamshouldertop
▽	hamstomach
¨	hamsymmlr
:	hamsymmpar
⌔	hamteeth
1	hamthumb
⌔	hamthumbball
1|	hamthumbside
⌔	hamtongue
⌔	hamtouch
⌔	hamunderchin
⌔	hamupperarm
⌔	hamwristback
{	hamaltbegin
}	hamaltend
,	hamcomma
!	hamexclaim
.	hamfullstop
|	hammetaalt
⍁	hammime
⍁	hamnondominant
я	hamnonipsi
?	hamquery
⌔	hamarcd
⌔	hamarcl
⌔	hamarcr
⌔	hamarcu
\	hambetween
⍁	hamcircled


hamcircledi


hamcircledl


hamcircledo


hamcircledr


hamcirclei


hamcircleil


hamcircleir


hamcirclel


hamcircleo


hamcircleol


hamcircleor


hamcircler


hamcircleu


hamcircleui


hamcircleul


hamcircleuo


hamcircleur


hamclockd


hamclockdl


hamclockdr


hamclockfull


hamclockl


hamclockr

hamclocku

hamclockul

hamclockur

hamdecreasing

hamellipseh

hamellipseul


hamellipseur


hamellipsev


hamfast


hamfingerplay

hamhalt


hamincreasing


hammoved


hammovedi


hammovedl


hammovedo


hammovedr

hammovei

hammoveil

hammoveir

hammovel

hammoveo

hammoveol

hammoveor

hammover

hammoveu

hammoveui

hammoveul

hammoveuo

hammoveur

hamnodding

hamnomotion

hamrepeatcontinue

hamrepeatcontinueseveral

hamrepeatfromstartseveral


hamrepeatreverse

hamreplace


hamrest

hamslow


hamstirccw


hamstircw


hamswinging


hamtense


hamtwisting

hamzigzag


hamarmextended


hambehind

hambrushing


hamcross

hamellipseh


hamellipseul


hamellipseur


hamellipsev

hamfusionbegin

hamfusionend


haminterlock


hamneutralspace

hamwavy

hamzigzag

ANEXO E. LISTA DE MOVIMIENTOS NO MANUALES

Muestra el contenido de cada archivo de texto que ha sido utilizado en la aplicación Editor para definir los movimientos de la cabeza, cejas, cuerpo, espalda, mirada, nariz y párpados. La primera columna representa el código para codificación SiGML, y la segunda columna la descripción del movimiento.

cap.txt

NO	Assentir (de dalt a baix)
SH	Negar (de esquerra a dreta)
SR	Girar cap a la dreta
SL	Girar cap a l'esquerra
TR	Inclinar cap a la dreta
TL	Inclinar cap a l'esquerra
NF	Inclinar cap endavant
NB	Inclinar cap enrere
PF	Moure cap endavant
PB	Moure cap enrere
LI	Moviment del cap vinculat amb la mirada
NU	Assentir un sol cop (de dalt fins al centre)
ND	Assentir un col cop (del centre fins a baix)

celles.txt

RB	Celles aixecades (ambdos)
RR	Cella dreta aixecada
RL	Cella esquerra aixecada
FU	Celles arrufades

cos.txt

RL	Girar cos a l'esquerra
RR	Girar cos a la dreta
TL	Inclinar cos a l'esquerra
TR	Inclinar cos a la dreta
TF	Inclinar cos endavant
TB	Inclinar cos enrere
SI	Sospirar
HE	Tirar (pit mogut cap amunt)
ST	Recte (tornar en vertical)
RD	Doblegar (Esquena doblegada)

esquena.txt

UL Espatlla esquerra aixecat
UR Espatlla dreta aixecat
UB Espatlles aixecades (ambdos)
HL Espatlla esquerra doblegada cap endavant
HR Espatlla dreta doblegada cap endavant
HB Espatlles doblegades cap endavant (ambdos)
SL Espatlla esquerra econgit (a dalt a baix)
SR Espatlla dreta econgit (a dalt a baix)
SB Espatlles encongides (ambdos)

mirada.txt

UP cap a dalt
DN cap a baix
LE cap a l'esquerra
RI cap a la dreta
LU cap a dalt esquerra
RU cap a dalt dreta
LD cap a baix esquerre
RD cap a baix dreta

nas.txt

WR Nas arrugat
TW Contraccions
WI Obrir fosses nasals

parpelles.txt

WB Parpelles obertes (amdos)
WR Parpella dreta oberta
WL Parpella esquerra oberta
SB Parpelles semi tancats (ambdos)
SR Parpella dreta semi tancat
SL Parpella esquerra semi tancat
CB Parpelles tancades (ambdos)
CR Parpella dreta tancat
CL Parpella esquerra tancat
TB Parpelles ben tancades (ambdos)
TR Parpella dreta ben tancada
TL Parpella esquerra ben tancada
BB Tancar i obrir els ulls

boca.txt

Posición	Código
Mejilla	C01 C02 C03 C04 C05 C06 C07 C08 C09 C10 C11 C12

Posición	Código
Labio	L01 L02 L03 L04 L05 L06 L07 L08 L09 L10 L11 L12 L13 L14 L15 L16 L17 L18 L19 L20 L21 L22 L23 L24 L25 L26 L27 L28 L29 L30 L31

Posición	Código
Lengua	T01 T02 T03 T04 T05 T06 T07 T08 T09 T10 T11 T12 T13 T14 T15 T16 T17

Posición	Código
Diente	D01 D02 D03 D04 D05 D06 D07 D08 D09

Posición	Código
Mandíbula	J01 J02